

# NASA Lab8 - docker & docker-compose

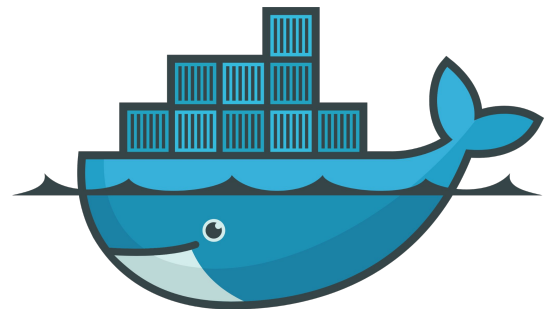
2025/04/14 TA 杜冠勳

# OUTLINE

- Introduction to Docker
- Why Docker?
- Docker Basic Concepts
- Virtualization Concepts: VM vs Container
  - VM Virtualization: Introduction to KVM, QEMU, and libvirt
  - Docker Container: Implementation on Non-Linux System and Linux System
- Overview of Docker Compose
- Docker Useful Command
- LAB TIME

# Introduction to Docker

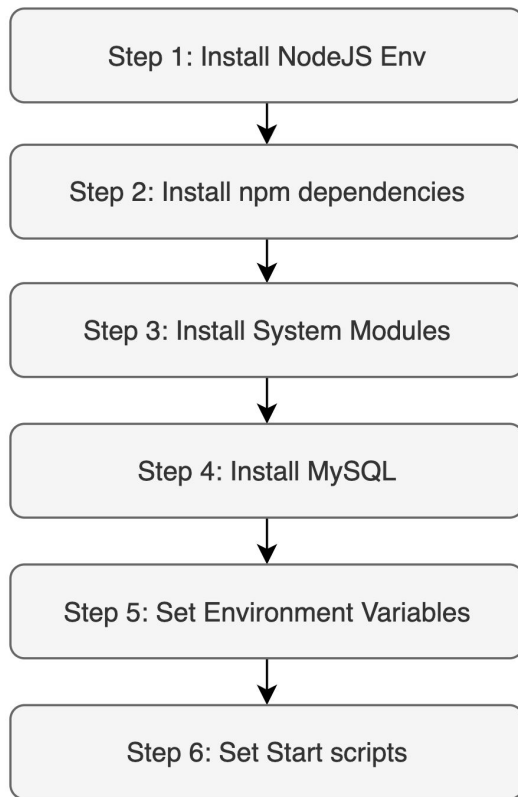
- Docker is a platform to build, run, and share applications.
- It lets us package an app **with all its dependencies**, so it runs easily on any machine.



docker

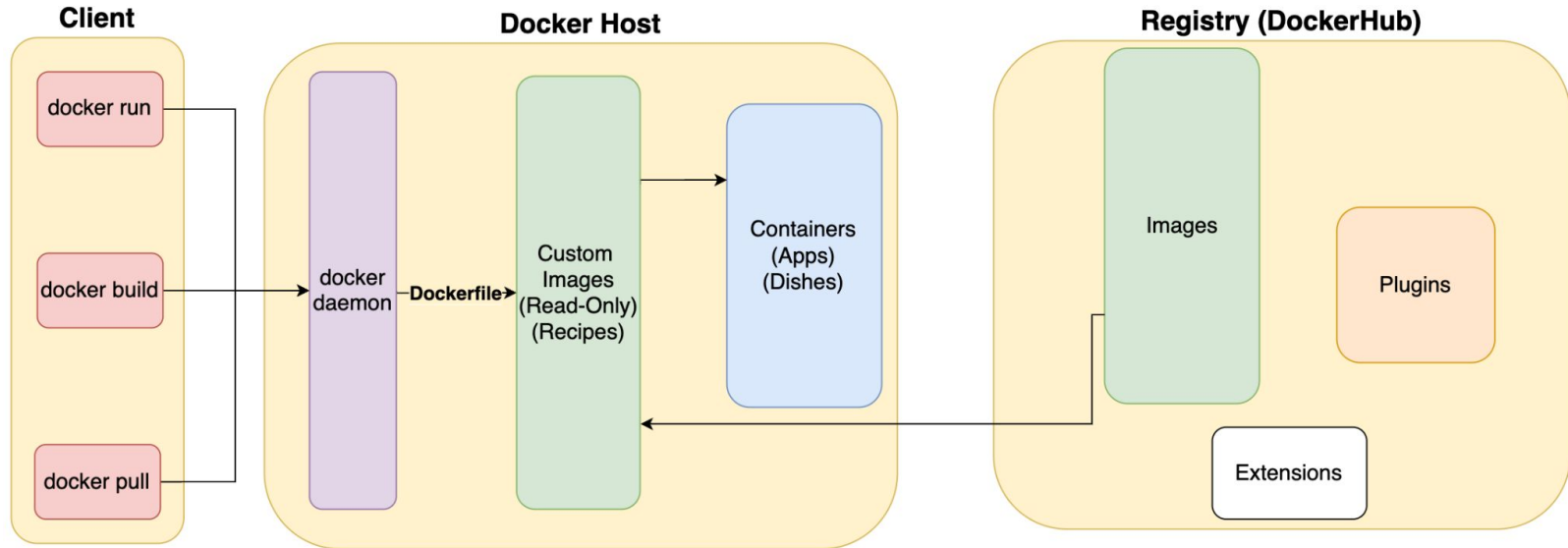
# Why Docker?

- Setting up apps with many components is time-consuming and painful.
- Docker avoids “dependency hell” by ensuring the app runs consistently across environments.



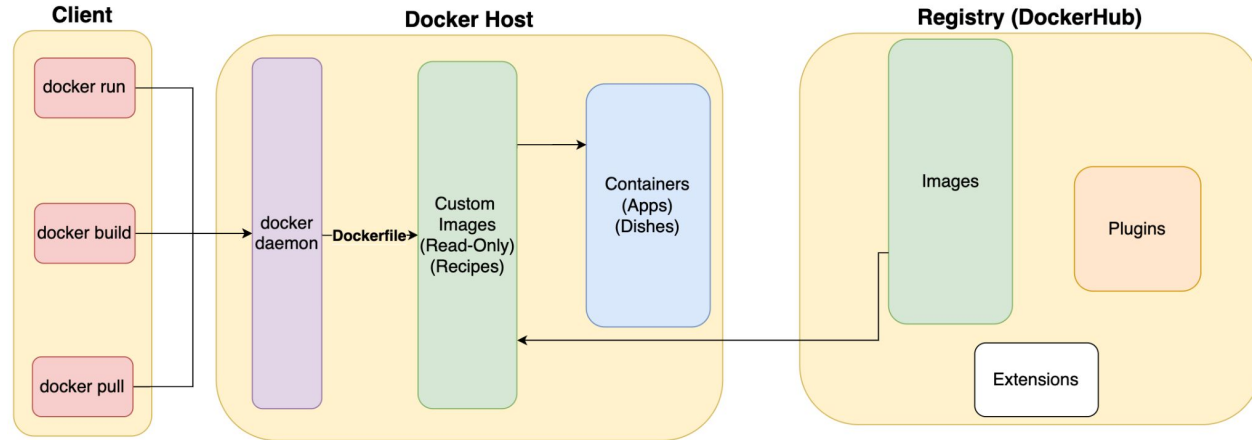
# Docker Basic Concepts

- Docker is a Server-Client structure.



# Docker Basic Concepts

- Docker Host: Server-side of Docker; our terminal is the Client.
- Registry: DockerHub stores and shares Docker images.
- Image: A read-only recipe used to create containers.
- We can use existing images or write a Dockerfile to build our own image.



# Virtualization: VM vs Docker Container

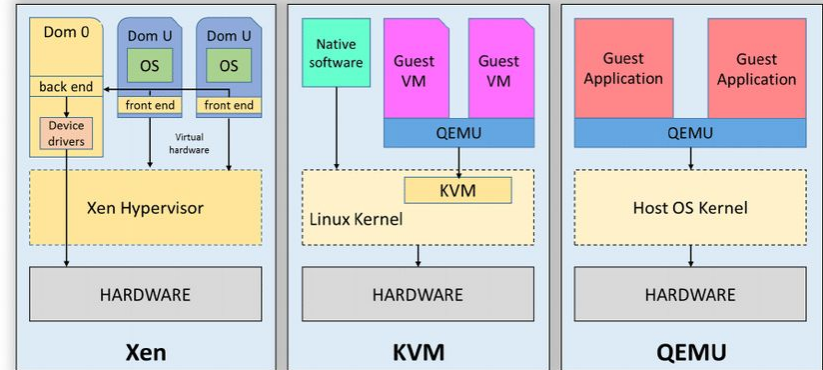
# VM Virtualization - KVM and QEMU

## How VMs Work: **Hypervisor**

- A hypervisor virtualizes hardware resources, allowing multiple virtual machines to run on one physical machine.

## Popular Hypervisor: QEMU + KVM

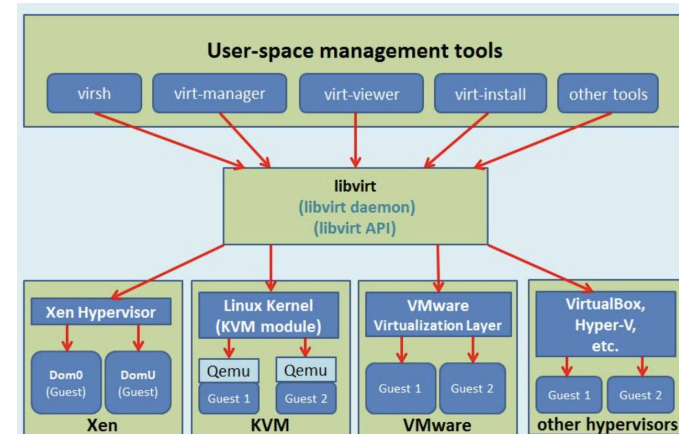
- QEMU: Emulates virtual hardware like memory, disk, and network.
- KVM: Built into the Linux kernel; allows VM code to run directly on hardware for better performance.





# VM Virtualization - libvirt

- Libvirt is an API layer that simplifies control of complex hypervisors.
- Supports multiple frontends: library (libvirt.so), CLI (virsh), GUI (virt-manager).
- Lets you manage VMs (start, stop, snapshots, network, storage) across different backends like QEMU/KVM or Xen.
- Uses XML configs to define VM specs, which are translated into low-level hypervisor commands.

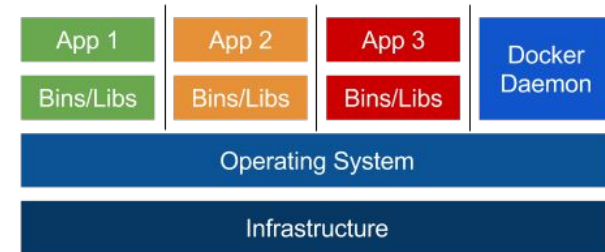
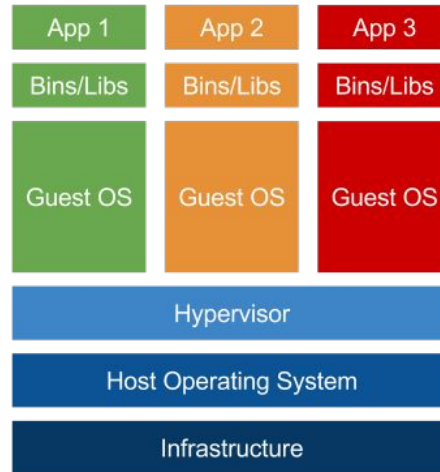


# Docker Virtualization - On Non-Linux System

- Docker Desktop **secretly runs a Linux VM** on your system (e.g., WSL<sub>2</sub> on Windows, Apple virtualization Framework on macOS).
- This is done because container implementation relies on certain features provided by the Linux kernel, such as namespaces, cgroups, and union file systems.
- Docker maps the VM's files and state to the host, so it feels like it's running natively.
- Since containers run in a Linux environment, they're portable across platforms, as long as the CPU architecture matches.

# Docker Virtualization - On Linux System

- Each container is essentially a lightweight, isolated process.
- The Docker Daemon is also just a complex Linux process.
- Docker uses native Linux features to give each container its own isolated environment.



# Overview of Docker Compose

- A tool for defining and running multi-container Docker applications.
- You describe your app's services in a YAML file, then start everything with a single command.
- It's like writing your docker commands in one place—making complex setups easy and repeatable.
- It feels like launching a full app stack as easily as starting a VM.

# Docker CLI Useful Command



## Cheatsheet for Docker CLI

### Run a new Container

Start a new Container from an Image

```
docker run IMAGE
docker run nginx
```

...and assign it a name

```
docker run --name CONTAINER IMAGE
docker run --name web nginx
```

...and map a port

```
docker run -p HOSTPORT:CONTAINERPORT IMAGE
docker run -p 8080:80 nginx
```

...and map all ports

```
docker run -P IMAGE
docker run -P nginx
```

...and start container in background

```
docker run -d IMAGE
docker run -d nginx
```

...and assign it a hostname

```
docker run --hostname HOSTNAME IMAGE
docker run --hostname srv nginx
```

...and add a dns entry

```
docker run --add-host HOSTNAME:IP IMAGE
```

...and map a local directory into the container

```
docker run -v HOSTDIR:TARGETDIR IMAGE
docker run -v ~/.usr/share/nginx/html nginx
```

...but change the entrypoint

```
docker run -it --entrypoint EXECUTABLE IMAGE
docker run -it --entrypoint bash nginx
```

### Manage Containers

Show a list of running containers

```
docker ps
```

Show a list of all containers

```
docker ps -a
```

Delete a container

```
docker rm CONTAINER
docker rm web
```

Delete a running container

```
docker rm -f CONTAINER
docker rm -f web
```

Delete stopped containers

```
docker container prune
```

Stop a running container

```
docker stop CONTAINER
docker stop web
```

Start a stopped container

```
docker start CONTAINER
docker start web
```

Copy a file from a container to the host

```
docker cp CONTAINER:SOURCE TARGET
docker cp web:/index.html index.html
```

Copy a file from the host to a container

```
docker cp TARGET CONTAINER:SOURCE
docker cp index.html web:/index.html
```

Start a shell inside a running container

```
docker exec -it CONTAINER EXECUTABLE
docker exec -it web bash
```

Rename a container

```
docker rename OLD_NAME NEW_NAME
docker rename 096 web
```

Create an image out of container

```
docker commit CONTAINER
docker commit web
```

### Manage Images

Download an image

```
docker pull IMAGE[:TAG]
docker pull nginx
```

Upload an image to a repository

```
docker push IMAGE
docker push myimage:1.0
```

Delete an image

```
docker rmi IMAGE
```

Show a list of all Images

```
docker images
```

Delete dangling images

```
docker image prune
```

Delete all unused images

```
docker image prune -a
```

Build an image from a Dockerfile

```
docker build DIRECTORY
docker build .
```

Tag an image

```
docker tag IMAGE NEWIMAGE
docker tag ubuntu ubuntu:18.04
```

Build and tag an image from a Dockerfile

```
docker build -t IMAGE DIRECTORY
docker build -t myimage .
```

Save an image to .tar file

```
docker save IMAGE > FILE
docker save nginx > nginx.tar
```

Load an image from a .tar file

```
docker load -i TARFILE
docker load -i nginx.tar
```

### Info & Stats

Show the logs of a container

```
docker logs CONTAINER
docker logs web
```

Show stats of running containers

```
docker stats
```

Show processes of container

```
docker top CONTAINER
docker top web
```

Show installed docker version

```
docker version
```

Get detailed info about an object

```
docker inspect NAME
docker inspect nginx
```

Show all modified files in container

```
docker diff CONTAINER
docker diff web
```

Show mapped ports of a container

```
docker port CONTAINER
docker port web
```

Ref:

<https://dockerlabs.collabnix.com/docker/cheatsheet/>

## Appendix: Other Docker-like Tools

- [OrbStack](#): macOS only, a lighter tool to host container, optimize the Linux VM running on macOS
- [Podman](#): Red Hat, a major company that sells Linux distributions for a living, replaced the Docker daemon with Podman.

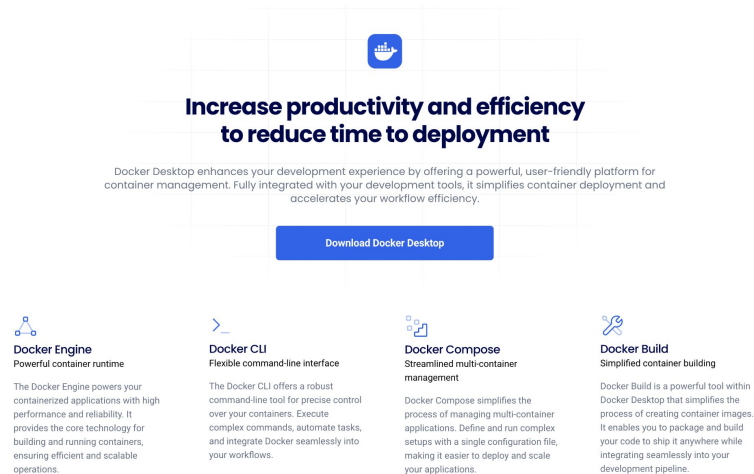
LAB TIME: Using Docker & Docker Compose to run  
DOMJudge

# Installation Method 1: Docker Desktop

This lab can be done on your local devices, no matter which OS you use.

If you use Docker Desktop, **remember to launch it each time**—this starts the Docker Server.


Download link: [Docker Desktop](#)



**Increase productivity and efficiency to reduce time to deployment**


Docker Desktop enhances your development experience by offering a powerful, user-friendly platform for container management. Fully integrated with your development tools, it simplifies container deployment and accelerates your workflow efficiency.

[Download Docker Desktop](#)




**Docker Engine**  
Powerful container runtime

The Docker Engine powers your containerized applications with high performance and reliability. It provides the core technology for building and running containers, ensuring efficient and scalable operations.




**Docker CLI**  
Flexible command-line interface

The Docker CLI offers a robust command-line tool for precise control over your containers. Execute complex commands, automate tasks, and integrate Docker seamlessly into your workflows.



**Docker Compose**  
Streamlined multi-container management

Docker Compose simplifies the process of managing multi-container applications. Define and run complex setups with a single configuration file, making it easier to deploy and scale your applications.



**Docker Build**  
Simplified container building

Docker Build is a powerful tool within Docker Desktop that simplifies the process of creating container images. It enables you to package and build your code to ship it anywhere while integrating seamlessly into your development pipeline.



# Installation Method 2: Docker Engine + Docker Compose

- [Tutorial link](#): This only works on Linux OS

```
# Tested On Ubuntu 24.04
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin
docker-compose-plugin

sudo usermod -aG docker $USER # logout + relogin
```

# Check Installation

```
docker run hello-world
```

```
docker version
```

```
~/LocalStorage/NASA/lab8
docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c9c5fd25a1bd: Pull complete
Digest: sha256:fc08e727181e2668370f47db6319815c279ed887e2f01be96b94106bc2781430
Status: Downloaded newer image for hello-world:latest
```

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.  
(arm64v8)
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/>

```
~/LocalStorage/NASA/lab8
```

```
docker version
```

```
Client:
Version:      28.0.4
API version:  1.48
Go version:   go1.23.7
Git commit:   b8034c0
Built:        Tue Mar 25 15:06:09 2025
OS/Arch:      darwin/arm64
Context:      desktop-linux
```

```
Server: Docker Desktop 4.40.0 (187762)
```

```
Engine:
Version:      28.0.4
API version:  1.48 (minimum version 1.24)
Go version:   go1.23.7
Git commit:   6430e49
Built:        Tue Mar 25 15:07:18 2025
OS/Arch:      linux/arm64
Experimental: false
containerd:
Version:      1.7.26
GitCommit:    753481ec61c7c8955a23d6ff7bc8e4daed455734
runc:
Version:      1.2.5
GitCommit:    v1.2.5-0-g59923ef
docker-init:
Version:      0.19.0
GitCommit:    de40ad0
```

# Run DOMserver + MariaDB Container

*# setup network*

```
docker network create domjudge-net
```

*# When using macOS ARM device, you can add --platform linux/amd64 to make Docker use QEMU simulate amd64.*

*# Please don't use special characters like % @ in your password.*

*# mariadb, -d is for running in background*

```
docker run -d --name dj-mariadb \  
  --network domjudge-net \  
  -e MYSQL_ROOT_PASSWORD=<ROOT_PASSWORD> \  
  -e MYSQL_USER=<USERNAME> \  
  -e MYSQL_PASSWORD=<PASSWORD> \  
  -e MYSQL_DATABASE=<DATABASE_NAME> \  
  -p 13306:3306 \  
  mariadb --max-connections=1000
```

*# domserver*

```
docker run -d --name domserver \  
  --network domjudge-net \  
  -e MYSQL_HOST=dj-mariadb \  
  -e MYSQL_USER=<USERNAME> \  
  -e MYSQL_DATABASE=<DATABASE_NAME> \  
  -e MYSQL_PASSWORD=<PASSWORD> \  
  -e MYSQL_ROOT_PASSWORD=<ROOT_PASSWORD> \  
  -p 12345:80 \  
  domjudge/domserver:latest
```

*# In http://localhost 12345*

The screenshot shows the DOMjudge web interface in a browser window. The address bar shows `http://localhost:12345/public`. The page title is "DOMjudge" with tabs for "Scoreboard" and "Problemset". A "Login" button is in the top right. The main content area is titled "Demo contest" with a timer showing "started: 04:43 -- ends: 09:43". Below the title is a "Filter" dropdown. The scoreboard table has columns: RANK, TEAM, SCORE, and three medal columns (A, B, C). The first row shows rank 1 for "Example teamname" (Utrecht University) with a score of 0/0. Below the table is a "Summary" row showing 0 solved problems. At the bottom left, there are two tables: "Cell colours" and "Medals (tentative)".

RANK	TEAM	SCORE	A	B	C
1	Example teamname Utrecht University	0 / 0			
Summary		0	0	0	0
			0	0	0
			0	0	0

Cell colours	Medals (tentative)
	Gold Medal
	Silver Medal
	Bronze Medal

Last Update: Thu 10 Apr 2025 07:00:00 CEST  
using DOMjudge

# Check status and Get Admin Password

```
docker ps -a # show all containers
```

```
# check logs
```

```
docker logs domserver
```

```
docker logs dj-mariadb
```

```
docker exec -it domserver bash
```

```
# Get Admin Password
```

```
cat /opt/domjudge/domserver/etc/initial_admin_password.secret
```

```
# Just showing
```

```
# It won't be used in this lab
```

```
cat /opt/domjudge/domserver/etc/restapi.secret
```

## Remove Containers

- First, take a screenshot of the results you just ran.
- Then, we're going to remove containers and redo it using Docker Compose.

```
docker stop domserver dj-mariadb
```

```
docker rm domserver dj-mariadb
```

# Do it again: Using docker-compose

*# You can find that you're basically just writing the parameters you'd use with docker run into a YAML file.*

*# docker-compose.yml*

```
services:
  mariadb:
    image: mariadb
    container_name: dj-mariadb
    environment:
      MYSQL_ROOT_PASSWORD: <ROOT_PASSWORD>
      MYSQL_USER: <USERNAME>
      MYSQL_PASSWORD: <PASSWORD>
      MYSQL_DATABASE: domjudge
    ports:
      - "13306:3306"
    networks:
      - domjudge-net
    command: --max-connections=1000
```

*# create and modify the file as left-handed*  
vim docker-compose.yml

```
domserver:
  image: domjudge/domserver:latest
  container_name: domserver
  depends_on:
    - mariadb
  environment:
    MYSQL_HOST: <MariaDB_container_name>
    MYSQL_USER: <USERNAME>
    MYSQL_DATABASE: domjudge
    MYSQL_PASSWORD: <PASSWORD>
    MYSQL_ROOT_PASSWORD: <ROOT_PASSWORD>
  ports:
    - "12345:80"
  networks:
    - domjudge-net
```

*# execute following commands*  
docker compose up -d

*# Use admin account to login domserver*  
*# Account: admin*  
*# Password: <PASSWORD\_YOU\_JUST\_GOT>*

```
networks:
  domjudge-net:
    driver: bridge
```

# Submission

**Please submit a single PDF file to [NTU COOL](#).**

The PDF must include:

1. A screenshot of the output from `docker ps -a`.
2. A terminal screenshot showing successful retrieval of the admin password.
3. Terminal screenshots of successful container startup using both methods:
  - a. Method 1: using `docker run`
  - b. Method 2: using `docker compose`
4. Your `docker-compose.yml` file.
5. A screenshot of a successful admin login to DOMJudge, with the URL showing `localhost:12345`.