CL2006 - Operating Systems Fall 2024

**LAB # 11MANUAL (Common)**

**Please note that all labs’ topics including pre-lab, in-lab and post-lab exercises are part of the theory and lab syllabus. These topics will be part of your Midterms and Final Exams of lab and theory.**

## Objectives:

1. **To understand and learn about implementing docker containers in Lab**

## Lab Tasks:

1. Compile and run the code workouts to familiarize yourself with docker.
2. Create your first website using the Django framework.

## Delivery of Lab contents:

## Strictly following the following content delivery strategy. Ask students to take notes during the lab.

## 1st Hour

## Understand about containers, Docker, dockerfile and YAML

## 2nd Hour

## Install dependencies, create a simple application in python

## 3rd Hour

## Push application to container and deploy it.

**Created by: Hamza Yousuf, January 2019 FAST-NU, Lahore Modified by: Nadeem Kafi (21/03/2024)**

**DEPARTMENT OF COMPUTER SCEICEN, FAST-NU, KARACHI**

**Background**

In recent years, containerization has emerged as a transformative technology in the field of software development and deployment. With its ability to streamline application packaging, isolation, and deployment, containers have become an essential tool for modern software development teams. In this comprehensive guide, we will delve into the theory behind containerization, explore its key concepts, and demonstrate practical examples through a hands-on lab session.

\*\*1.1 What are Containers?\*\*

Containers are lightweight, portable, and self-contained units that package applications and their dependencies. Unlike traditional virtual machines, which require a separate operating system instance for each application, containers share the host operating system's kernel while providing isolated environments for applications.

\*\*1.2 Historical Context\*\*

The concept of containerization traces its roots back to UNIX chroot jails and FreeBSD jails. However, it gained widespread adoption with the development of Linux Containers (LXC) and later, Docker, which revolutionized container technology by providing user-friendly tools for container management.

Key Concepts of Containerization\*\*

\*\*2.1 Namespaces\*\*

Namespaces provide process isolation within a container by creating separate namespaces for various system resources such as processes, networks, filesystems, and more. This isolation ensures that processes within a container are unaware of processes outside the container, enhancing security and resource management.

\*\*2.2 Control Groups (cgroups)\*\*

Control groups, or cgroups, allow fine-grained control over resource allocation and usage within containers. By limiting CPU, memory, disk I/O, and other resources, cgroups ensure that containers operate within specified resource constraints, preventing resource contention and ensuring consistent performance.

\*\*2.3 Docker and Container Orchestration\*\*

Docker, initially released in 2013, played a significant role in popularizing containerization by providing a user-friendly interface for building, running, and managing containers. Container orchestration tools such as Kubernetes further streamline container deployment and management at scale, automating tasks such as scaling, load balancing, and service discovery.

**Practical Lab Session: Install Dependencies**

 **Windows**: Download and install Docker Desktop.

 **Mac**: Use Docker Desktop.

 **Linux**: Use your package manager:

**For Linux:**

**sudo apt update**

**sudo apt install docker.io**

**Verify installation:**

docker –version

A close up of a number

Description automatically generated with medium confidence

**Basic Docker Commands**

Docker contains a plethora of commands, all serving a different purpose. Since we haven’t created a container, yet we will utilize a built-in image provided by docker.

**Run a container:**docker run hello-world

A screenshot of a computer

Description automatically generated

To see how many containers are running we can use the following command

docker ps

A screenshot of a computer

Description automatically generated

To stop a container, we can use the following command.  
docker stop <container-id>

**Note: The ps command displays all active containers, to display inactive containers use the -a flag.**

To remove a container, use the following command,

docker rm <container-id>   
  
**Note: Active containers would restrict users to being deleted, first stop the container then remove it.**

**Building a Docker Image:**

A **Docker image** is a lightweight, standalone, and executable software package that includes everything needed to run a piece of software: the code, runtime, libraries, environment variables, and configurations. It serves as a blueprint to create **Docker containers**.

**Key Features of a Docker Image**

1. **Layers**: Images are built in layers, where each layer represents an instruction in the Dockerfile (e.g., FROM, RUN, COPY).
2. **Portability**: They can be shared via repositories like Docker Hub.
3. **Versioning**: Images are immutable; changes result in a new version.

Create a simple Dockerfile: (Note: The file name is dockerfile with no extension in front of it)

# Use an official base image

FROM python:3.9-slim

# Set working directory

WORKDIR /app

# Copy files

COPY . .

# Install dependencies

RUN pip install flask

# Expose port

EXPOSE 5000

# Command to run the app

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

We can compile the docker file now but we have no application pushed onto the container.  
So to test the container we would create a python app where it display a web page using the flask api.

**Introduction to Flask**

**What is Flask?**  
Flask is a lightweight Python framework for building web applications.  
(See Here: <https://flask.palletsprojects.com/en/stable/>)

For the purposes of this tutorial lets just create a simple app

Create a file app.py

**from flask import Flask**

**app = Flask(\_\_name\_\_)**

**@app.route('/')**

**def home():**

**return "Hello, Docker!"**

**if \_\_name\_\_ == '\_\_main\_\_':**

**app.run(host='0.0.0.0', port=5000)**

Then run the app locally to see if it works

python app.py (Some of you might not have an older python version so use python3 instead of python)

A screenshot of a computer

Description automatically generated

Now our app is running we can close the server and push the application to the docker container.

**Deploying Flask on Docker:**

1. Create the following structure:

/docker-test

├── app.py

└── dockerfile

1. Build the Docker image:

docker build -t docker-test .

A screenshot of a computer program

Description automatically generated

1. Run the container:

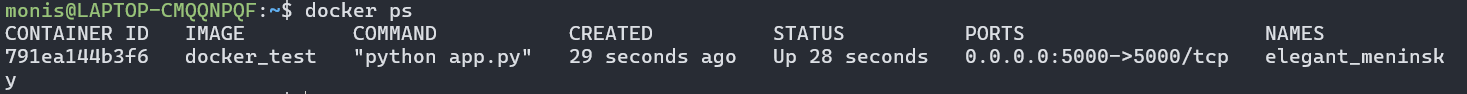
docker run -p 5000:5000 docker-test

A screenshot of a computer

Description automatically generated

1. Open http://localhost:5000 to see the app. (Note: Or use 127.0.0.1:5000)

Open another terminal to see the container running



Application:

A screenshot of a computer

Description automatically generated

Task:   
Create a Web application of your choice where the admin can assign a voting poll, and the user can vote on it. Deploy it on a docker container and display it using the following commands. (Use whatever framework to design the web page).