TDDC88/TDDC93: Software Engineering

Lab 1
DOCKER

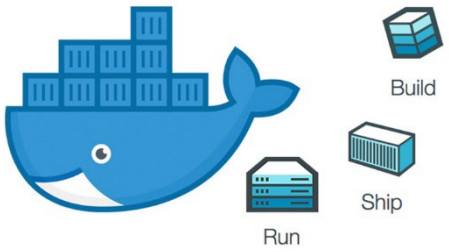
Purpose:

- To give you a fundamental understanding and practical experience with Docker
- To learn some base commands for the Docker CLI
- To get familiar with how to build your own Docker images
- To understand why it is said that the main slogan of Docker is: build an application, ship it anywhere, and run it anywhere

Docker

We recommend the following sources of information to start with:

- https://docs.docker.com/get-started/
- https://docs.docker.com/docker-hub/



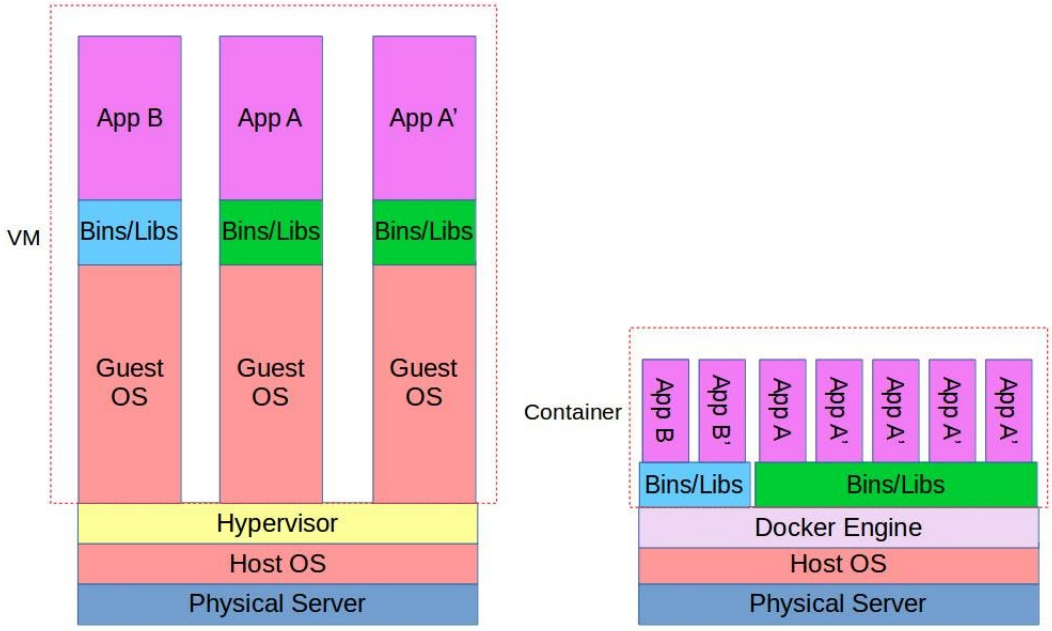
Introduction:

1. What is docker?

The container movement is driven by Docker company. The main intent of the Docker project is to allow developers to create, deploy, and run applications easier by the use of containers. Their main slogan is building an application, ship it anywhere, and run it anywhere. Containerization is a method for operating system virtualization. It allows you to run an application and its dependencies isolatedly. Container allows you to easily package an application's code, dependencies, and configurations into handy building blocks. Containers can help to ensure that irrespective of the deployment environment, applications deploy quickly, reliably, and consistently.

2. What is the difference between a container and a virtual machine?

Virtual machines give isolation at the hardware abstraction layer, whereas containers provide OS-level process isolation. You can see the difference in the following figure obviously:



As it is visible from this figure, containers are isolated but share the host OS, whereas a unique OS is running within each VM. The two technologies can be used together for added advantages. For instance, a container can be created inside a virtual machine in order to make a solution ultra-portable.

3. Docker Terminology

3.1. Docker Engine

In the *official documents of Docker*, Docker Engines is described as follows [1]:

"*Docker Engine acts as a client-server application with:*

- *A server with a long-running daemon process (the* docker *command).*
- *APIs which specify interfaces that programs can use to talk to and instruct the Docker daemon.*
- *.A command line interface (CLI) client (the* docker *command).*"

Daemon runs on the host machine to create and manage docker objects such as images, containers, and networks. In order to control or interact with the Docker daemon through scripting or direct CLI commands, the CLI uses the Docker REST API.

3.2. Docker registry

Docker registry is the storage component for Docker images. You can store images in either a

public or private repositories. Based on [2], "Docker Hub *is a cloud-based registry service which allows you to link to code repositories, build your images and test them, stores manually pushed images, and links to Docker Cloud so you can deploy images to your hosts. It provides a centralized resource for container image discovery, distribution and change management,* user and team collaboration*, and workflow automation throughout the development pipeline.*" There are many image repositories that you can find them here: https://hub.docker.com/

3.3. Docker images and Docker containers

These two concepts are closely related. You can create images from a *Dockerfile* with the docker build command. Images are stored in Docker registries, such as Docker Hub. You should notice that Docker images themselves are never "started" and never "running". The docker run command grabs a Docker image as a template and produces a container from it.

The goal of the lab

The goal of this lab is to give you a fundamental understanding and practical experience of Docker containers. In order to achieve this, there are several parts in this lab, which you should go through them and answer the given questions.

How to run Docker machine on IDA machines

For those who want to run Docker images on lab computers, please run the following commands to run the Docker machine:

```
cd /courses/TDDI41/TDDC88/
./start_docker_machine.sh
```

Note: when you start the docker_machine, a new pop-up QEMU terminal will be started and you must wait until it finishes the processing, do not proceed further until you see a message in this terminal "docker login" (it might take some time for the machine to start). Now switch back to your original terminal, you get a message like "Your machine up. Access it with ssh student@127.0.0.01 -p 2220". Enter the given ssh command (ssh student@127.0.0.01 -p 2220) in your original terminal with given password (which is password) to access the Docker machine. You will see that you are entered in the Docker machine.

Part 1. Hello world

In this part, you learn how to run a container. To run a new container, you should use docker run <image_name> command. To test it, enter the following command in your terminal.

```
docker run hello-world
```

Q1. Write the answers to the following questions and show it to your instructor.

1) What do you see as the output?
2) What is your conclusion from the printed messages as output?
3) How is the image found?

Part 2. Docker CLI

In this part, you are going to pull an Ubuntu image from Docker hub. You can pull any public image from Docker hub.

2.1. In your terminal, enter the following two commands. Save this information somewhere for yourself.

```
ls
uname -a
```

Q2. What each of the above commands does?

2.2. Use the following command to run Ubuntu image.

```
docker run -it ubuntu /bin/bash
```

After typing this command, first it is tried to find the Ubuntu locally and if Docker cannot find it, looks for it remotely from the Docker hub and pull it, and then runs this container locally. Now you are running an Ubuntu on top of your computer and you made it to run /bin/bash command. If you wanted to pull the image but not to run it you could use `docker pull <image_name>` command.

Q3. Now enter the two mentioned commands in the previous section (2.1) again in this container and compare the results with the previous results. What is your conclusion?

Now enter "exit" command so you can exit form Ubuntu container.

2.3. You can use following commands to list images. Do you see any available images? Write about it.

```
docker images
```

2.4. In order to list running containers, use the following command:

```
docker ps
```

2.5. Now if you want to see all containers you can use -a switch as follow:

```
docker ps -a
```

Q4. Is Ubuntu still in the list of containers? What is the difference between docker ps and docker ps -a?

2.6. You can use the following command to remove Ubuntu form the list:

```
docker rm <id>
```

Q5. What is the difference between docker rm and docker rmi commands?

Part 3. Your first image

Note. In the case that you do not know how to create and modify your files, you have to get familiar with editors such as vi. There are many online cheat sheets for these editors.

In this part, you get familiar with building your own Docker images. Docker images can be built automatically by reading from a *Dockerfile* [3]. All the commands a user could call on the command line to assemble an image should be written in a text document called *Dockerfile*. There are two different main types of Docker images: child images and base images. Most Dockerfiles start from a parent image. However, you might need to create a base image if you need to control the contents of your image completely.

- Child images: images that are created based on a parent image. It refers to the contents of the FROM directive in the Dockerfile. Each subsequent declaration in the Dockerfile modifies this parent image.
- Base image: images that have FROM scratch in their Dockerfile.

So far, if you wanted to start developing a Python application, your first step was installing a Python runtime on your machine. In this part, you learn with Docker you can have a portable Python runtime as an image, and no installation is needed! In this part, we use the example given in [4] with small modifications. The goal is in this part is to create an image that sand-boxes a simple Flask application. In which, a random cat.gif is displayed every time it is loaded. In the following steps, you are going to create your three needed files to build this image: app.py, requirements.txt, and Dockerfile.

Step1: app.py

you should create a folder and get inside it:

```
mkdir pyapp

cd pyapp
```

Then, there are two ways to have the application app.py in this folder (you can find it here: https://github.com/prakhar1989/docker-curriculum/blob/master/flask-app/app.py): (A) download it (using wget command) or (B) create it yourself.

(A) Download app.py:

You should use the raw url of app.py to download it:

```
wget https://raw.githubusercontent.com/prakhar1989/docker-curriculum/master/flask-app/app.py
```

(B) Create app.py:

The first step is to create app.py file as follows:

```
touch app.py
```

Open it with your favourite text editor and add the following content (you can find it here: https://github.com/prakhar1989/docker-curriculum/blob/master/flask-app/app.py):

```
from flask import Flask, render_template
import os
import random


app = Flask(__name__)
# list of cat images
images = [
    "https://firebasestorage.googleapis.com/v0/b/docker-curriculum.appspot.com/o/catnip%2F0.gif?alt=media&token=0fff4b31-b3d8-44fb-be39-723f040e57fb",
    "https://firebasestorage.googleapis.com/v0/b/docker-curriculum.appspot.com/o/catnip%2F1.gif?alt=media&token=2328c855-572f-4a10-af8c-23a6e1db574c",
    "https://firebasestorage.googleapis.com/v0/b/docker-curriculum.appspot.com/o/catnip%2F10.gif?alt=media&token=647fd422-c8d1-4879-af3e-fea695da79b2",
    "https://firebasestorage.googleapis.com/v0/b/docker-curriculum.appspot.com/o/catnip%2F11.gif?alt=media&token=900cce1f-55c0-4e02-80c6-ee587d1e9b6e",
```

```
    "https://firebasestorage.googleapis.com/v0/b/docker-curriculum.appspot.com/o/catnip%2F2.gif?alt=media&token=8a108bd4-8dfc-4dbc-9b8c-0db0e626f65b",

    "https://firebasestorage.googleapis.com/v0/b/docker-curriculum.appspot.com/o/catnip%2F3.gif?alt=media&token=4e270d85-0be3-4048-99bd-696ece8070ea",

    "https://firebasestorage.googleapis.com/v0/b/docker-curriculum.appspot.com/o/catnip%2F4.gif?alt=media&token=e7daf297-e615-4dfc-aa19-bee959204774",

    "https://firebasestorage.googleapis.com/v0/b/docker-curriculum.appspot.com/o/catnip%2F5.gif?alt=media&token=a8e472e6-94da-45f9-aab8-d51ec499e5ed",

    "https://firebasestorage.googleapis.com/v0/b/docker-curriculum.appspot.com/o/catnip%2F7.gif?alt=media&token=9e449089-9f94-4002-a92a-3e44c6bd18a9",

    "https://firebasestorage.googleapis.com/v0/b/docker-curriculum.appspot.com/o/catnip%2F8.gif?alt=media&token=80a48714-7aaa-45fa-a36b-a7653dc3292b",

    "https://firebasestorage.googleapis.com/v0/b/docker-curriculum.appspot.com/o/catnip%2F9.gif?alt=media&token=a57a1c71-a8af-4170-8fee-bfe11809f0b3",
]

@app.route("/")
def index():
    url = random.choice(images)
    return render_template("index.html", url=url)


if __name__ == "__main__":
    app.run(host="0.0.0.0", port=int(os.environ.get("PORT", 5000)))
```

In app.py, Flask looks for templates in the templates folder, which you should provide for it. So, while you are inside pyapp folder, create a folder called templates and download index.html inside it:

```
mkdir templates
cd templates
wget https://raw.githubusercontent.com/prakhar1989/docker-curriculum/master/flask-app/templates/index.html
```

Then go back to pyapp folder:

```
cd ..
```

Step2: requirements.txt

Similar to app.py, download or create a file named requirements.txt.

(A) Download requirements.txt:

```
wget https://raw.githubusercontent.com/prakhar1989/docker-curriculum/master/flask-app/requirements.txt
```

(B) Create requirements.txt:

Create a file named requirements.txt, then open it and add:

```
Flask== 2.0.3
```

Step 3: Dockerfile

Finally, download or create Dockerfile:

(A) Download Dockerfile:

```
wget https://raw.githubusercontent.com/prakhar1989/docker-curriculum/master/flask-app/Dockerfile
```

(B) Create Dockerfile:

Create a file name Dockerfile and add:

```
FROM python:3

WORKDIR /usr/src/app

COPY . .

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

EXPOSE 5000
```

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

By reading Docker manuals, try to understand what each line of this Dockerfile means.

Q1. What each line of this Dockerfile mean?

Step4: Build image

Now we have the Dockerfile and can build your first image. In order to build your images, enter:

```
docker build -t cats .
```

Q2. Do you build a base or child image? If you are creating a child image, explain which modifications are made to the parent image.

Step5: Run image

The last step in this part is to run the image and see if it works:

```
docker run -p 4000:5000 cats
```

Q3. What does -p 4000:5000 do?

If you have done everything correctly, by browsing localhost:4000 in your browser, you should see a cat image. You can share your image with your friend easily by pushing it to a Docker registry such as Docker Hub. By creating an account on Docker Hub and using `docker push` command, you can share your images with others. If you are interested to see more details about sharing images visit https://docs.docker.com/get-started/part3/.

After finishing this example, you should be able to explain what you have done to your lab assistant and answer the given questions.

Examination

You should understand everything in each part deeply to be able to answer the questions that assistants ask you. Furthermore, you have to answer all questions. Contact your assistant during a lab session and be ready to answer questions concerning what you did when demonstrating. You don't need to hand in anything.

Refrences

[1] https://docs.docker.com/get-started/overview/
[2] https://docs.docker.com/docker-hub/
[3] lhttps://docs.docker.com/engine/reference/builder/
[4] https://docker-curriculum.com/#our-first-image