COMS 7900: Docker Usage Instruction

Yousaf Khaliq, Haoyuan Wang, Danlei Zhu

February 2025

Docker is an open-source platform that allows developers to build, package, and run applications in lightweight, portable containers. It simplifies application deployment by ensuring that software runs consistently across different environments. Users can pull image from Dockerhub or build a image, and then run container build on image.

1 Introduction

Docker usage involves two important definitions:

- Image: A container image is a static file that contains all the code, libraries, dependencies, and configurations needed to run an application. It is like a recipe or blueprint. It contains everything your application needs—code, libraries, settings—to run. It's a packaged, ready-to-use snapshot.
- Container: A container is a running instance of an image. It provides an isolated, lightweight environment where your application executes exactly as defined in the image. It's what you get when you "cook" that recipe. It's a running instance of the image, an isolated environment where your application runs exactly as described in the image.

2 Files and commands

All files can be found at https://github.com/danleezhu/Docker-usage/tree/main

2.1 Files needed

- **Application (Programming) File:**
  This file contains your code—your functions or main application logic—that performs the desired tasks. It is the core program that the container will run.
- **requirements.txt:**
  This file lists all the external packages and libraries required for your application. It ensures that the container has the necessary dependencies to run your program.
- **Dockerfile:**
  This file defines the instructions for building a Docker image. It specifies the base operating system, environment variables, dependencies, and commands needed to create the environment in which your application will run.
- **docker-compose.yml:**
  This file orchestrates multi-container Docker applications. It defines how your Docker image(s) are built and run, including service configuration, networking, and volume mappings.

2.2 Commands for building image

1. When files are ready to go, we will build the image named by `Dockerfile` using command:

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

2. We could pack our built image into zip .tar file by:

   ```
   docker save -o my_container.tar image_name
   ```

3. Tag our image:

   ```
   docker tag image_dockerhub_username/image_name
   ```

4. Push tagged image into dockerhub

   ```
   docker push dockerhub_username/image_name
   ```

3 Singularity usage in Windows and iOS

3.1 Windows

Docker hub with WSL 2 should be installed and Ubuntu enabled (Settings - Resources - WSL integration). Singularity should also already be installed.

1. Access the terminal in Docker hub and start WSL and Ubuntu by running the following command

   ```
   wsl -d Ubuntu
   ```

2. Save and export the Docker image to a .tar file (if not done already)

   ```
   docker save -o my_tarfile_name.tar Docker-usage:latest
   ```

3. Import the .tar file into Singularity

   ```
   singularity build my_sif_filename.sif docker-archive://my_tarfile_name.tar
   ```

4. Run the Singularity Container

   ```
   singularity run my_sif_filename.sif
   ```

3.2 iOS

Using below code needs to replace `<container_id>` by real container ports/id; replace `docker_username` by your real dockerhub account name; replace `image_name` by what you defined when building the image.

3.2.1 Method 1: leverage ubuntu container

When we try to install Singularity (now called Apptainer) on macOS, we run into the lack of support for Singularity on macOS. This is mainly because Apptainer relies on Linux-specific features such as namespaces and cgroups. Therefore, in response to this problem, the following is a possible operation process for creating a sif file in macOS:

1. Start a Docker container. There are many options such as Ubuntu.

   ```
   docker pull ubuntu:latest
   docker run -dit ubuntu
   ```

2. Upload tar file to this Docker container.

   ```
   docker cp my_container.tar <container_id>:/home
   ```

3. If the container cannot use singularity, you need to install it. The command is as follows:

   ```
   apt-get update
   apt-get install -y singularity-container
   ```

4. After that, you can create sif file in this container.

   ```
   singularity build my_container.sif docker-archive://my_container.tar
   ```

5. Finally, transfer sif file to local machine. (Recommand using `docker cp` command)

   ```
   docker cp <container_id>:/home/my_container.sif  /Desktop/my_container.sif
   ```

6. (Optional) Other packages might be needed by different macbook, it depends on specific fatal information. We have one macbook faced an invalid path issue, which could be solved by:

   ```
   apt-get update && apt-get install -y tzdata
   rm -f /etc/localtime
   cp /usr/share/zoneinfo/UTC /etc/localtime
   ls -l /etc/localtime
   ```

7. Run singularity:

   ```
   singularity run my_container.sif
   ```

8. Exit ubuntu container:

   ```
   exit()
   ```

3.2.2 Method 2: leverage `Multipass`

Use multipass, install and conduct singularity in a virtual machine environment. It required to use image tagged and pushed into Dockerhub.

1. install command multipass

   ```
   brew install --cask multipass
   ```

2. Check multpass version:

   ```
   multipass version
   ```

3. Lauch virtual machine:

   ```
   multipass launch --name singularity-vm
   ```

4. Open and go in interactive vm:

   ```
   multipass shell singularity-vm
   ```

5. Get update:

   ```
   sudo apt update
   ```

6. Install singularity-container in vm:

   ```
   sudo apt install -y singularity-container
   ```

7. Install other useful packages that might need, avoid errors:

   ```
   sudo apt install -y build-essential libseccomp-dev pkg-config squashfs-tools cryptsetup
   uidmap wget git
   ```

8. Use singilarity build .SIF file from reading image built and pushed into dockerhub:

   ```
   singularity build my_container.sif docker://dockerhub_username/image_name
   ```

9. Run singularity:

   ```
   singularity run my_container.sif
   ```

10. Exit vm after task finished:

    ```
    exit
    ```

11. Step and delete vm created before:

    ```
    multipass list
    multipass stop singularity-vm
    multipass delete singularity-vm
    multipass purge
    ```