

## Docker (Task A)

### Task A

#### [Github Repository](#)

#### A-1: Demonstrate ability to write simple Dockerfile

Dockerfile is a text configuration file written using a special syntax describing step-by-step instructions of all the commands needed to assemble a Docker image. The **docker build** command processes this file generating a Docker Image in the Local Image Cache which can then be start-up using the **docker run** command, or pushed to a permanent Image Repository.

Here is a list of (a few) Dockerfile Commands (Full list can be found [here](#))

<b>ADD</b>	Defines files to copy from the Host file system onto the Containe
<b>CMD</b>	This is the command that will run when the Container starts
<b>ENTRYPOINT</b>	Sets the default application used every time a Container is created from the Image. If used in conjunction with CMD, you can remove the application and just define the arguments there
<b>ENV</b>	Set/modify the environment variables within Containers created from the Image
<b>EXPOSE</b>	Define which Container ports to expose
<b>FROM</b>	Select the base image to build the new image on top of
<b>LABEL</b>	Optional field to let you identify yourself as the maintainer of this image. This is just a label (it used to be a dedicated Docker directive).
<b>RUN</b>	Specify commands to make changes to your Image and subsequently the Containers started from this Image. This includes updating packages, installing software, adding users, creating an initial database, setting up certificates, etc. These are the commands you would run at the command line to install and configure your application. This is one of the most important dockerfile directives.
<b>COPY</b>	This will copy the files/folders from the source to the destination in the container

For Task A, we only require FROM and COPY Dockerfile commands. We can use the Nginx base image found in Docker Hub and copy our html files into the destination folder in the container as well as the **nginx.conf** file to replace the **default.conf** from the Nginx base image.

The Dockerfile can be found in my Github Repository

## A-2: Setup Nginx and run a reverse proxy

### What is a proxy server?

A proxy server is a go-between or intermediary server that forwards requests for content from multiple clients to different servers across the Internet. A reverse proxy server is a type of proxy server that typically sits behind the firewall in a private network and directs client requests to the appropriate backend server. A reverse proxy provides an additional level of abstraction and control to ensure the smooth flow of network traffic between clients and servers.

The main reason for creating a reverse proxy is so that one can host everything under one domain name or ip address under port 80 and don't require the user specify special port numbers when making requests to the frontend, backend or other services.

For example, we are avoiding the following:

- Domain Name: `http://exampledomain.com`
- Frontend: `http://exampledomain.com:3001`
- Backend: `http://exampledomain.com:5000`

What we want to achieve is the following:

- Domain Name: `http://exampledomain.com`
- Frontend: `http://exampledomain.com`
- Backend: `http://exampledomain.com/demo`

To setup Nginx and run the reverse proxy, we need to write a `nginx.conf` to replace the `default.conf` file. The changes can be seen in the Github Repository. Follow the instructions in README.md for a demo of the reverse proxy set up.

### A-3: Demonstrate ability to use Docker Hub registry to keep track of images

- To demonstrate the ability of using Docker Hub registry to keep track of images, we can build our own image on top of images that are already present.
- In this case, we will build upon a base **ubuntu** image and install Git software on this container instance.
- After installing Git on the container instance of the base ubuntu image, we need to save that state of the container as an image, so that we can relaunch additional new containers from that image. We thus commit the modified container image locally and then push to the Repository Hub's account's Registry.

```
ASUS@JackChen MINGW64 /c/Program Files/Docker Toolbox
$ docker run -it --name JackContainer ubuntu:latest
root@00f013d4ff16:/# apt-get update
root@00f013d4ff16:/# git --version
git version 2.25.1
root@00f013d4ff16:/#
```

We can then commit the container **JackContainer** to my Repository Hub's account's Repository by executing **docker commit JackContainer jackimaru/ubuntu-git** we can then push the image to the Docker Hub using **docker push jackimaru/ubuntu-git**

```
ASUS@JackChen MINGW64 /c/Program Files/Docker Toolbox
$ docker push jackimaru/ubuntu-git
The push refers to repository [docker.io/jackimaru/ubuntu-git]
2d437332186d: Pushed
a4399aeb9a0e: Mounted from library/ubuntu
35a91a75d24b: Mounted from library/ubuntu
ad44aa179b33: Mounted from library/ubuntu
2ce3c188c38d: Mounted from library/ubuntu
latest: digest: sha256:82cea601d931d94f67d4523c4a0a6cf2d1524b175dd35fd1e843f4077cdd8008 size: 1364
```



jackimaru [Edit profile](#)

Community User Joined September 9, 2020

[Repositories](#)

[Starred](#)

[Contributed](#)

Displaying 1 of 1 repository



jackimaru/ubuntu-git

By [jackimaru](#) • Updated 2 minutes ago

Container

1 0  
Download Stars


When we want to get the container from Docker Hub registry to local machine, we can simply execute the command **docker pull jackimaru/ubuntu-git** in the docker quickstart terminal.

The image with the reverse proxy is also uploaded [here](#). A screenshot is shown below.

Docker Hub

hub.docker.com/repository/docker/jackimaru/taska


We've updated our Terms of Service. [Learn more.](#)

 Search for great content (e.g., mysql)


ExploreRepositoriesOrganizationsGet Helpjackimaru

Repositoriesjackimaru / taskaUsing 0 of 1 private repositories. [Get more](#)

GeneralTagsBuildsTimelineCollaboratorsWebhooksSettings

 **jackimaru / taska**  
*This repository does not have a description*  
Last pushed: a few seconds ago

**Docker commands** [Public View](#)  
To push a new tag to this repository,  
`docker push jackimaru/taska:tagname`

**Tags**  
This repository contains 1 tag(s).  
latest  a few seconds ago  
[See all](#)

**Recent builds**  
*Link a source provider and run a build to see build results here.*

## Appendix: Individual Learning about Docker

### What is Docker?

- Docker comes into play during the **Deployment** stage
- It helps make the process of application deployment easy and efficient and resolves issues related to deploying applications
- Due to the Software stack:
  - Frontend components
  - Backend workers
  - DB
  - Env/lib dependencies
- Some system may not be able to run the software stack
- Docker is a tool designed to make it easier to deploy and run applications by using containers
- Containers allow developer to package an application with all the part it needs, such as libraries and other dependencies, and ship it all out as one package
- Allows for **separation of concern**, allowing the developer to focus on coding and developing and not to worry about deployment
- CI/CD: Consistently test and deploy code to different environments e.g. Stage, UAT, Production

### Terminology

In the last section, you saw a lot of Docker-specific jargon which might be confusing to some. So before you go further, let's clarify some terminology that is used frequently in the Docker ecosystem.

- **Images** - The file system and configuration of our application which are used to create containers. To find out more about a Docker image, run `docker inspect alpine`. In the demo above, you used the `docker pull` command to download the alpine image. When you executed the command `docker run hello-world`, it also did a `docker pull` behind the scenes to download the hello-world image. Images are **immutable**
- **Containers** - Running instances of Docker images — containers run the actual applications. A container includes an application and all of its dependencies. It shares the kernel with other containers, and runs as an isolated process in user space on the host OS. You created a container using `docker run` which you did using the alpine image that you downloaded. A list of running containers can be seen using the `docker ps` command. Containers are Ephemeral
- **Docker daemon** - The background service running on the host that manages building, running and distributing Docker containers.
- **Docker client** - The command line tool that allows the user to interact with the Docker daemon.
- **Docker Store** - A registry of Docker images, where you can find trusted and enterprise ready containers, plugins, and Docker editions. You'll be using this later in this tutorial

## General Development Workflow

- Create and test individual containers for each component of the application by first creating Docker images
- Assemble the containers and supporting infrastructure into a complete application
- Test, share and deploy the complete containerized application

## Learning about Image and Containers

- Docker image captures the private filesystem that the containerized process will run in. An image is needed to be created that contains just what the application needs to run
- Dockerfile describes how to assemble a private filesystem for a container, and also can contain some metadata describing how to run a container based on this image.
- Images can exist without containers, whereas a container needs to run an image to exist. Therefore, containers are dependent on images and use them to construct a run-time environment and run an application

## Commands:

- **docker pull httpd**
  - Only downloads the Docker image to local machine and does not run it
- **docker run <image-name>**
  - Tries to find the image locally first and runs a container (create an instance of the image)
  - If not found, it will pull the image from the Docker hub, which is similar to downloading it and could take a while to do that depending on the internet connection
  - Once pulled successfully, it is present in the local repository and hence will be able to create a container based on this image.
- **docker ps [-all]**
  - Gives a list of all running containers [show history of containers launched]

```
C:\Users\ASUS\Desktop\CS3219_Assignment\A\node-bulletin-board\bulletin-board-app>docker ps -all
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
f0d949a16187	busybox	"sh"	9 minutes ago	Exited (0) 8 minutes ago		hungry_hellman

- **CONTAINER\_ID** : A Unique ID for the container that was launched.
- **IMAGE** : This was the IMAGE that was launched i.e. busybox
- **COMMAND** : Important. This was the default command that was executed when the container was launched. If you recollect, when the container based on busybox image was launched, it led you to the Unix Prompt i.e. the Shell was launched. And that is exactly what the program in /bin/sh does. This should give you a hint that in case you want to package your own Server in a Docker image, your default command here would typically be the command to launch the Server and put it in a listening mode.

- **docker images**

- Gives list of images present on the docker setup locally

```
C:\Users\ASUS\Desktop\CS3219_Assignment\A\node-bulletin-board\bulletin-board-app>docker images
```

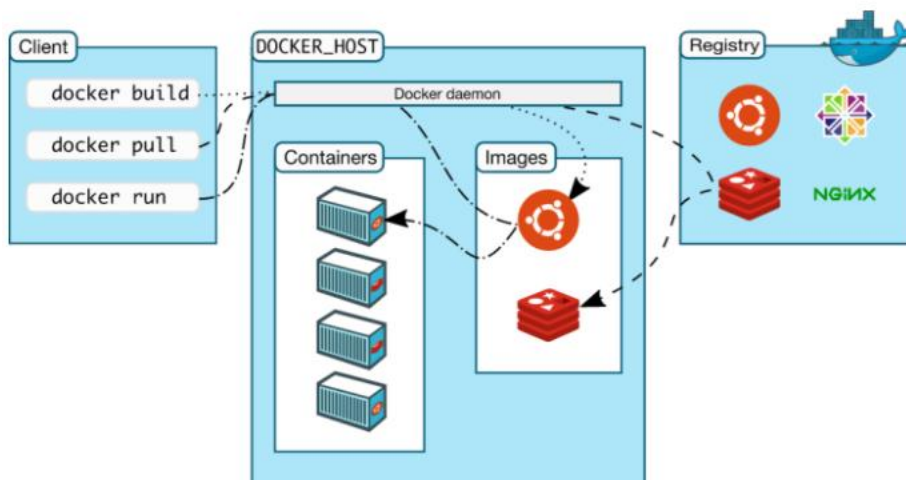
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
bulletinboard	1.0	eaef1be963b	22 minutes ago	183MB
busybox	latest	6858809bf669	5 hours ago	1.23MB
node	current-slim	05f62d57259e	11 days ago	167MB
hello-world	latest	bf756fb1ae65	8 months ago	13.3kB

- **REPOSITORY** column is obvious as it is the name of the Image itself

- **TAG** is important as it provides the version of the Image as there could be multiple versions of an image present in the Docker Hub
- Can mention the version TAG as needed: `docker run -t -i busybox:1.0`
- `docker start <CONTAINER_ID>`
  - Starts a stopped container
- `docker attach ab0e37214358`
  - Attach to a running Container via the docker attach command.
- `docker run -v [/VolumeName] --name <ContainerName> <ImageName>`
  - Mounting a volume for container
  - Data volumes are initialized when container is created. They can be shared across containers. They can be mounted in read-only mode as well.
- `docker-machine ip default`
  - Find the hostname on the command line

## Docker Hub

- Docker toolset consists of:
  - Docker Daemon
  - Docker Client
  - Docker Hub
- The Docker Architecture diagram is as follows:



- 
- the Registry is also called the Hub
- Docker hosts public repositories called the Docker Registry (Hub) a list of public Docker images can be found and used. This is handy, since a lot of developers have worked hard to get the images ready for us and all we need to do is pull those images and start launching containers based on them

## Data Volume

- A data volume is a specially designed directory in the container.
- It is initialized when the container is created. By default, it is not deleted when the container is stopped. It is not even garbage collected when there is no container referencing the volume.
- The data volumes are independently updated. Data volumes can be shared across containers too. They could be mounted in read-only mode too.

## Dockerfile

- There are several commands supported like **FROM**, **CMD**, **ENTRYPOINT**, **VOLUME**, **ENV** and more. We shall look at some of them
- Overall flow
  - Create a Dockerfile with the required instructions
  - Use the docker build command to create a Docker image based on the Dockerfile that was created in step 1