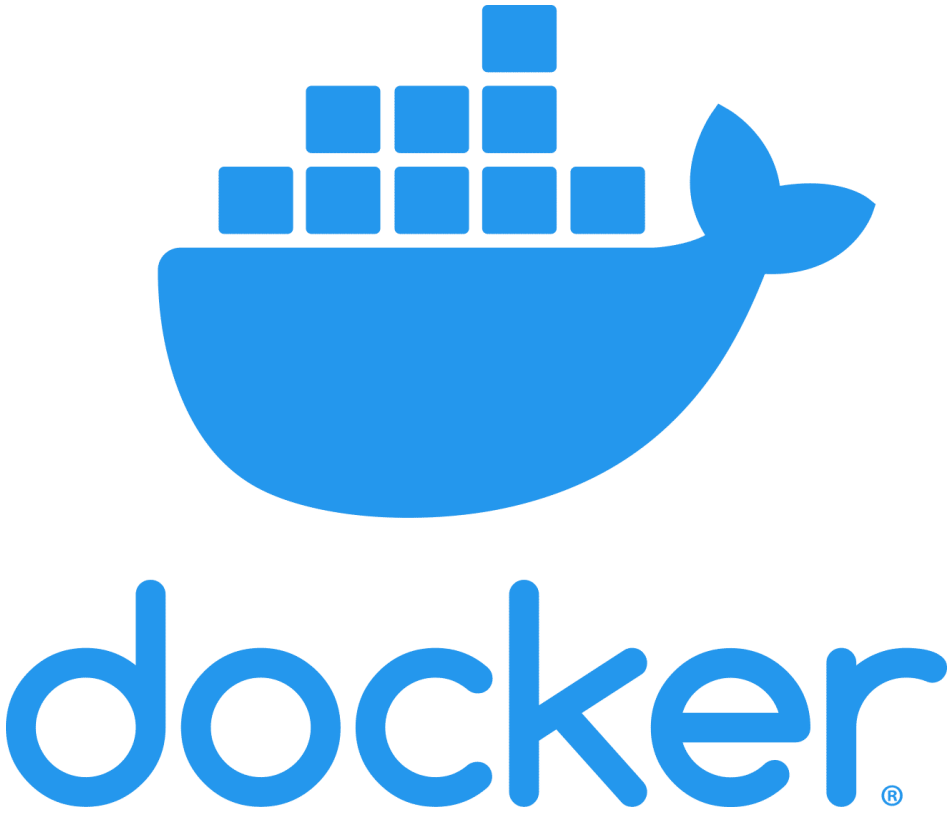
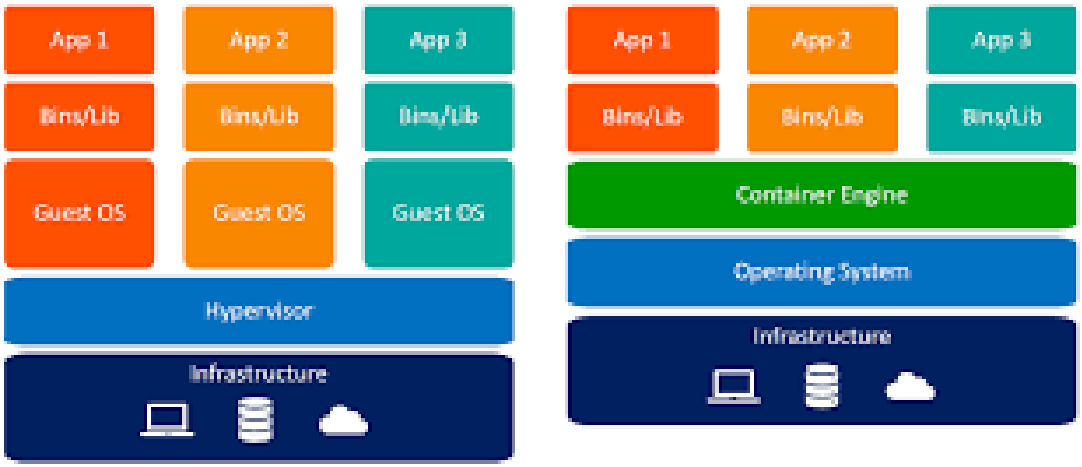
Containerization



Containerization

- A Docker image is made up of a collection of files that bundle together all the essentials – such as installations, application code, and dependencies – required to configure a fully operational container environment.
- You can create a Docker image by using one of two methods:
  - **Interactive**: By running a container from an existing Docker image, manually changing that container environment through a series of live steps, and saving the resulting state as a new image.
  - **Dockerfile**: By constructing a plain-text file, known as a Dockerfile, which provides the specifications for creating a Docker image.



Virtual Machines Containers

Credit: https://jfrog.com/knowledge-base/a-beginners-guide-to-understanding-and-building-docker-images/

Image Layers

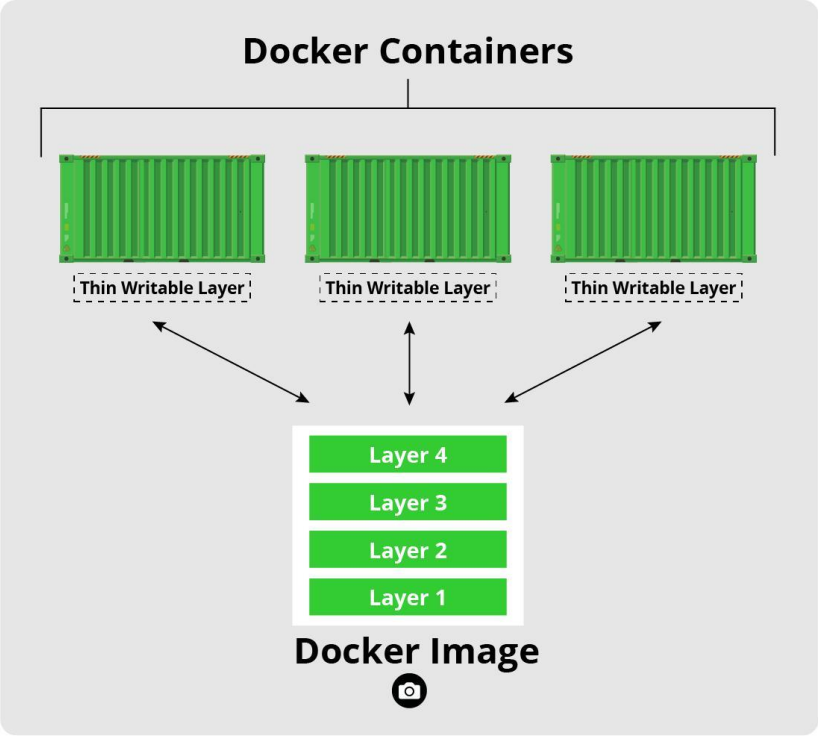
- **A Docker image** is a file used to execute code in a Docker container. Docker images act as a set of instructions to build a Docker container, **like a template.** Docker images also act as the starting point when using Docker. An image is comparable to a snapshot in virtual machine (VM) environments.
- Each of the files that make up a Docker image is known as a **layer**.
- These layers form a series of intermediate images, built one on top of the other in **stages**, where each layer is dependent on the layer immediately below it.
- when you make changes to a layer in your image, Docker not only rebuilds that particular layer, but all layers built from it. Therefore, a change to a layer at the top of a stack involves the least amount of computational work to rebuild the entire image.

- **Base Image:** a base image is an empty first layer, which allows you to build your Docker images from scratch.
  - Base images give you full control over the contents of images
  - are generally intended for more advanced Docker users.

Credit: https://jfrog.com/knowledge-base/a-beginners-guide-to-understanding-and-building-docker-images/

Container Layer

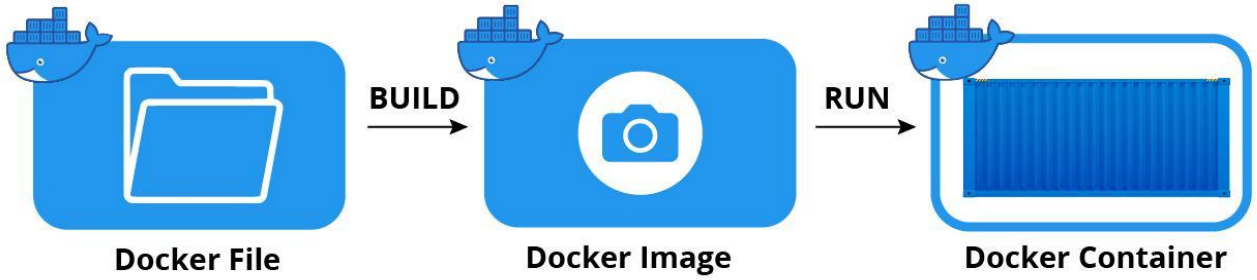
- **A Docker container** is a virtualized runtime environment used in application development.
- Each time Docker launches a container from an image, it adds a thin writable layer, known as the container layer, which stores all changes to the container throughout its runtime.
- As this layer is the only difference between a live operational container and the source Docker image itself, any number of like-for-like containers can potentially share access to the same underlying image while maintaining their own individual state.



Credit: https://jfrog.com/knowledge-base/a-beginners-guide-to-understanding-and-building-docker-images/

Docker Manifest

- Together with a set of individual layer files, a Docker image also includes an additional file known as a manifest. This is essentially a description of the image in JSON format and comprises information such as image tags, a digital signature, and details on how to configure the container for different types of host platforms.



Credit: https://jfrog.com/knowledge-base/a-beginners-guide-to-understanding-and-building-docker-images/

Install Docker

- **Check you machine compatibility**
  - **windows**
  - **Linux**
- Follow Installation steps in this official documentation
- For Ubuntu 20.04 follow this link
- After installation check the docker services are running:
  - **sudo systemctl status docker**

Output
● docker.service - Docker Application Container Engine Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled) Active: **active (running) since** Tue 2020-05-19 17:00:41 UTC; 17s ago TriggeredBy: ● docker.socket Docs: https://docs.docker.com Main PID: 24321 (dockerd) Tasks: 8 Memory: 46.4M CGroup: /system.slice/docker.service └─24321 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Credit: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04

Start Use Docker

- **Run docker command**

Output

```
  attach     Attach local standard input, output, and error streams to a running container
  build      Build an image from a Dockerfile
  commit     Create a new image from a container's changes
  cp         Copy files/folders between a container and the local filesystem
  create     Create a new container
  diff       Inspect changes to files or directories on a container's filesystem
  events     Get real time events from the server
  exec       Run a command in a running container
  export     Export a container's filesystem as a tar archive
  history    Show the history of an image
  images     List images
  import     Import the contents from a tarball to create a filesystem image
  info       Display system-wide information
  inspect    Return low-level information on Docker objects
  kill       Kill one or more running containers
  load       Load an image from a tar archive or STDIN
……
```

Credit: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04

Run First Docker Container

- Run command **docker run hello-world**

Output
Unable to find image 'hello-world:latest' locally latest: **Pulling from library/hello-world** 0e03bdcc26d7: Pull complete Digest: sha256:6a65f928fb91fcfbc963f7aa6d57c8eeb426ad9a20c7ee045538ef34847f44f1 Status: Downloaded newer image for hello-world:latest Hello from Docker! This message shows that your installation appears to be working correctly.

Credit: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04

Pull Docker Image

- Run command **docker pull ubuntu**

Output
Using default tag: latest
latest: **Pulling from library/ubuntu**
d51af753c3d3: Pull complete
fc878cd0a91c: Pull complete
6154df8ff988: Pull complete
fee5db0ff82f: Pull complete
Digest: sha256:747d2dbbaaee995098c9792d99bd333c6783ce56150d1b11e333bbceed5c54d7
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest

Credit: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04

List Docker Images

- Run command **docker images**

Output

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	latest	1d622ef86b13	3 weeks ago	73.9MB
hello-world	latest	bf756fb1ae65	4 months ago	13.3kB

Credit: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04

Running a Docker Container

- Run command **docker run -it ubuntu**

Output

root@d9b100f2f636:/#

Run in Bash Mode

- docker run -it ubuntu **/bin/bash**
  - This **allows a running container to create or modify files and directories in its local filesystem**.

Credit: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04

Detach a running container

- Press **Ctrl-P, followed by Ctrl-Q**, to detach from your connection. You'll be dropped back into the shell but the previously attached process will remain alive.
- Press **Ctrl-P, followed by Ctrl-D**, to detach from your connection. You'll be dropped back into the shell and the container is stopped.

Credit: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04

List Containers

- Run command **docker ps -a**

Output

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
1c08a7a0d0e4	ubuntu	"/bin/bash"	2 minutes ago	Exited (0) 8 seconds ago		quizzical_mcnulty
a707221a5f6c	hello-world	"/hello"	6 minutes ago	Exited (0) 6 minutes ago		youthful_curie

Credit: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04

Start/Stop Containers

- Run command docker start 1c08a7a0d0e4 docker stop 1c08a7a0d0e4

Output

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

1c08a7a0d0e4 ubuntu "/bin/bash" 2 minutes ago Exited (0) 8 seconds ago quizzical_mcnulty

a707221a5f6c hello-world "/hello" 6 minutes ago Exited (0) 6 minutes ago youthful_curie

Credit: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04

Copy files from/to Container

- docker **cp** <src-path> <container>:<dest-path>
- docker **cp** <container>:<src-path> <local-dest-path>

Credit: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04

Build Docker Image From File

- Deploy simple Python web application using flask
  - Create Simple **flask** web server: dockerize.py

```
from flask import Flask
app = Flask(__name__)
@app.route("/")
def hello():
  return "Hello World!"
if __name__ == "__main__":
  app.run(host="0.0.0.0", port=int("5000"), debug=True)
```

  - Create Python Requirements file **requirements.txt**

```
flask
```

  - Create Docker File **Dockerfile, COPY** command will copy current directory content to the folder **app** inside the container

```
FROM python:alpine3.7
COPY .  /app
WORKDIR /app
RUN pip install -r requirements.txt
EXPOSE 5000
CMD python ./dockerize.py
```

Build Docker Image From File

- Build docker image
  - docker **build** --tag flask-docker .
- Run Docker image and **publish** the port
  - docker **run** -p 5000:5000 **flask-docker**

Expose Vs Publish

1. If you specify neither EXPOSE nor -p, the service in the container will only be accessible from *inside* the container itself.
2. If you EXPOSE a port, the service in the container is not accessible from outside Docker, but from inside other Docker containers. So this is good for **inter-container communication**.
3. If you EXPOSE and -p a port, the service in the container is accessible from anywhere, even outside Docker.
4. If you do -p, but do not EXPOSE, Docker does an implicit EXPOSE. This is because if a port is open to the public, it is automatically also open to other Docker containers. Hence -p includes EXPOSE. This is effectively same as 3).

Save Container as image

- Save as image: docker **commit** 1c08a7a0d0e4 <img_tag>
- To test run: docker **images**

Exercise 01: build Nginx server docker container file

Nginx is a web server that can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache.

- Create Docker File **Dockerfile**

```
# Pull the minimal Ubuntu image
FROM ubuntu
# Install Nginx
RUN apt-get -y update && apt-get -y install nginx
# Copy the Nginx config
COPY default /etc/nginx/sites-available/default
# Expose the port for access
EXPOSE 80/tcp
# Run the Nginx server
CMD ["/usr/sbin/nginx", "-g", "daemon off;"]
```

- Create Ngnix Server Default Config file**: default**

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    root /usr/share/nginx/html;
    index index.html index.htm;
    server_name _;
    location / {
        try_files $uri $uri/ =404;
    }}
```

- **Building the Image:** $ docker build -t ngnix/server .
- **Run:** docker run -p 80:80 ngnix/server