Assignment 02

## Q1: Explain Docker Containers vs VMs

**VM:** A virtual machine (VM) is an emulation of a computer system. Put simply, it makes it possible to run what appear to be many separate computers on hardware that is actually one computer.

**Benefits of VMs**

* All OS resources available to apps
* Established management tools
* Established security tools
* Better known security controls

**Popular VM Providers**

* VMware vSphere
* VirtualBox
* Xen
* Hyper-V
* KVM

**Containers:** With containers, instead of virtualizing the underlying computer like a virtual machine (VM), just the OS is virtualized. Containers sit on top of a physical server and its host OS — typically Linux or Windows. Each container shares the host OS kernel and, usually, the binaries and libraries, too. Shared components are read-only. Sharing OS resources such as libraries significantly reduces the need to reproduce the operating system code, and means that a server can run multiple workloads with a single operating system installation. Containers are thus exceptionally light — they are only megabytes in size and take just seconds to start. Compared to containers, VMs take minutes to run and are an order of magnitude larger than an equivalent container.

**Benefits of Containers**

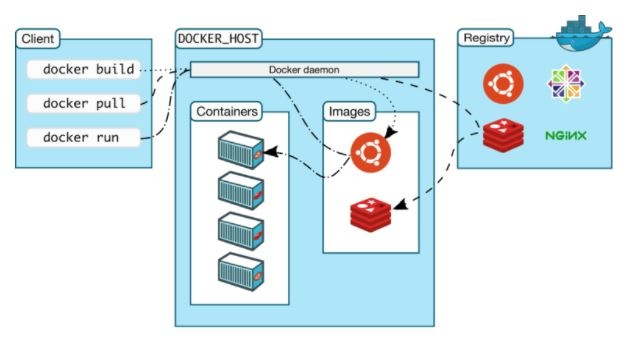
* Reduced IT management resources
* Reduced size of snapshots
* Quicker spinning up apps
* Reduced & simplified security updates
* Less code to transfer, migrate, upload workloads

**Popular Container Providers**

* Linux Containers
* LXC
* LXD
* CGManager
* Docker
* Windows Server Containers

## Q2: Explain Docker Architecture

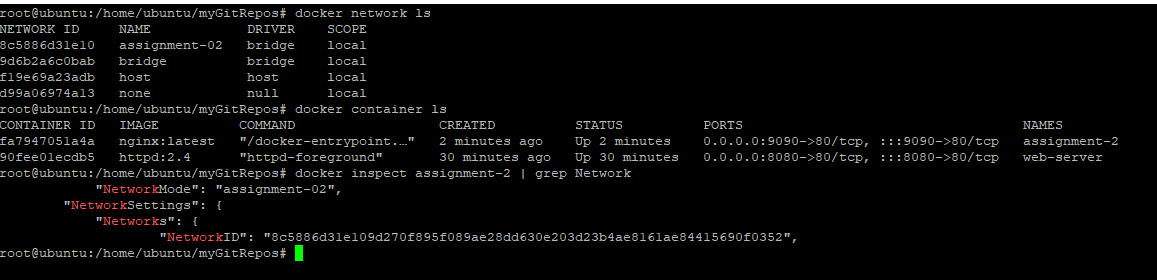
Docker’s architecture is also client-server based. However, it’s a little more complicated than a virtual machine because of the features involved. It consists of four main parts:



1. Docker Client: This is how you interact with your containers. Call it the user interface for Docker.
2. Docker Objects: These are your main components of Docker: your containers and images. We mentioned already that containers are the placeholders for your software, and can be read and written to. Container images are read-only, and used to create new containers.
3. Docker Daemon: A background process responsible for receiving commands and passing them to the containers via command line.
4. Docker Registry: Commonly known as Docker Hub, this is where your container images are stored and retrieved.

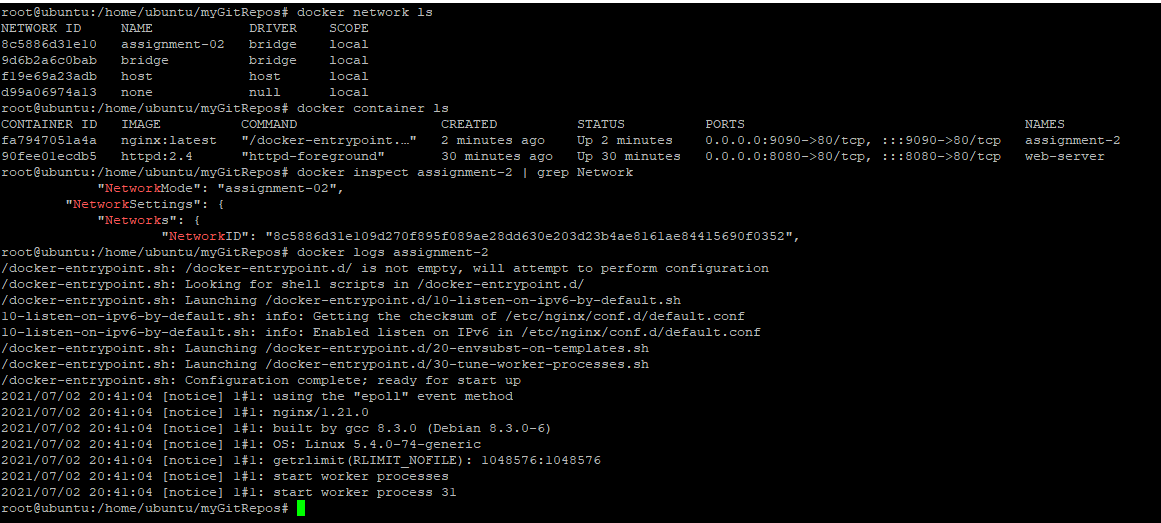
## Q3: Write command to create an nginx container in detached mode with name assignment-2 running on host port 9090 on a custom network named assignment-2

$ docker run -d --name assignment-2 --network assignment-02 --publish 9090:80 nginx:latest



## Q4: Write command to see logs of the above container

$ docker logs assignment-2

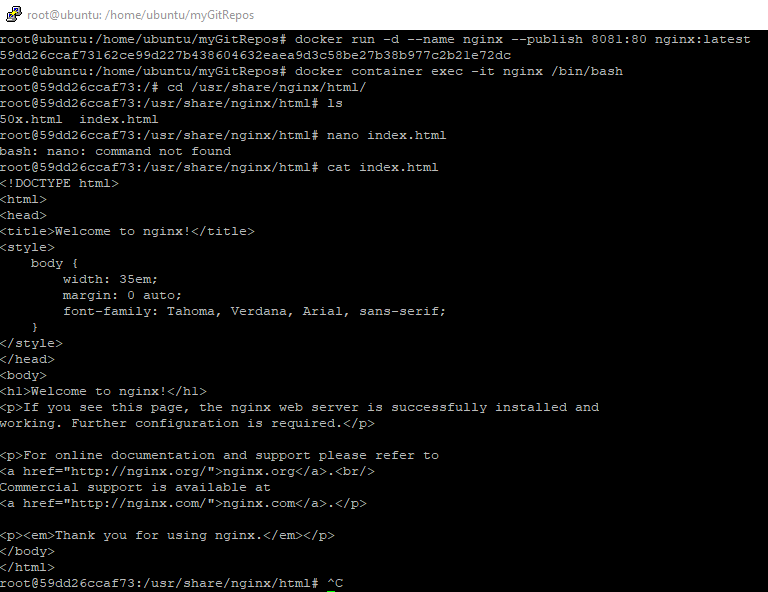


Q5: Write commands to Exec into the container and cat the output of the default nginx file at /usr/share/nginx/html/index.html

# docker run -d --name nginx --publish 8081:80 nginx:latest

# docker container exec -it nginx /bin/bash

# cat /usr/share/nginx/html/index.html



## Q6 &7: Exit the above container, and now recreate the container by Volume using bind mounting; then Command to exec into the above container and replace the default index.html to a custom one, which says that “I am becoming a Docker Expert” and it should be persisted for the next times

# docker run -dit --name web-server -p 8080:80 -v /tmp/nginx/html/:/usr/local/apache2/htdocs/ httpd:2.4

# nano /tmp/nginx/html/index.html

