GLOSSARY

Docker is:

- an open platform, written in Go language, for sharing, shipping, and running apps;
- allows to separate apps from our infrastructure to deliver software quickly;
- we can package and run an app within a container which provides security and isolation to run many containers on a host;
- manages the lifecycle of containers

Container is:

- loosely isolated environment;
- a runnable instance of an image; defined by its image as well as configuration options provided when we created/started it; if container removed, any changes to its state, not stored in persistent storage, disappear;
- lightweight since it does not need a hypervisor; it runs directly within the kernel of host machine; for this reason, there is the possibility to run more containers than virtual machines (VMs) on given hardware combination; it can be run within host machines that are actually VMs; relatively well isolated from other containers and its host machine;
- using Docker API or CLI we can create, start, stop, move or delete a container;
- containers can be connected to one or more networks, attach storage to it or create a new image based on its current state;
- the use of containers to deploy applications is called containerisation

  - containerisation is increasingly popular because containers are flexible, lightweight, portable, loosely coupled, scalable, secure;

Virtual Machine (VM) resembles a physical machine since it has CPU, memory, disk to store files, can connect to Internet if needed, however, it exists only as code, a software defined computer running on top of a hypervisor; a software that allows to run OS within another OS (e.g. VirtualBox, VMware)

Hypervisor is a software that exists between a physical hardware and the VM; ensures each VM receives the resources it needs in an orderly manner (e.g. in case of Google Cloud, VMs are running in the cloud)

Cloud is a network of powerful servers running hypervisors in data centres all over the world; the Internet is used to access information and services running on the cloud from app, browser, device;

Docker Engine is a client-server application with the following components:

- server which is a long-running program called a daemon process (`dockerd` command)
- REST API specifying interfaces that programs can use to talk to the daemon and instruct it what to do
- a command line interface (CLI) client (the `docker` command) which uses the REST API to control/interact with the docker daemon through scripting or direct CLI commands

Docker daemon (`dockerd`) listens for Docker API requests and manages Docker objects; can communicate with other daemons to manage Docker services

Docker client (`docker`) is the primary way Docker users interact with Docker; the client can send commands to `dockerd` to carry them out using the command docker run; `docker` command uses the Docker API; the client can communicate with more than one daemon;

Docker registries = stores Docker images; e.g. Docker Hub is a public registry that anyone can use and Docker looks for images on Docker Hub by default; the commands `docker pull` or `docker run` pull the required images from the configured registry; the command `docker push` pushes the image to our configured registry

Docker objects such as images, containers, networks, plugins, volumes, etc.

Image is a read-only template/package with instructions for creating a Docker container; often an image is based on another image with additional customisation; we can either create our own image or use images created by others published in a registry;

- to build our own image we create a *Dockerfile*, with a simple syntax for defining the steps needed to create the image and run it; each instruction in *Dockerfile* creates a layer in the image; when changing and rebuilding the *Dockerfile*, only those layers which have changed are rebuilt; all these make the images lightweight, small and fast when compared to other virtualisation technologies

Namespaces = a technology providing isolated workspace called the container; when running a container, Docker creates a set of namespaces for that container that provide a layer of isolation; each aspect of a container runs in a separate namespace and its access is limited to that namespace;

Docker Engine uses the following namespaces:

- the `pid` namespace: process isolation (PID: Process ID)
- the net namespace: managing network interfaces (NET: Networking)
- the `ipc` namespace: managing access to IPC resources (IPC: InterProcess Communication)
- the `mnt` namespace: managing filesystem mount points (MNT: Mount)
- the `uts` namespace: isolating kernel and version identifiers (UTS: Unix Timesharing System)

Control groups, a technology `cgroup` limiting an app to a specific set of resources; cgroups allow Docker Engine to share available hardware resources to containers and optionally enforce limits and constraints (e.g limiting the memory available to a specific container)

Union file systems or UnionFS are file systems that operate by creating layers making them lightweight and fast; Docker Engine uses UnionFS to provide the building blocks for containers;

Container Format is a wrapper/combination of namespaces, control groups and UnionFS. Default container format is `libcontainer`;

Instructions used in tutorial for Dockerfile

FROM sets the Base Image for subsequent instructions.
ADD copies new files, directories or remote file to container.
RUN execute any commands in a new layer on top of the current image and commit the results.
ENTRYPOINT configures a container that will run as an executable.

REFERENCES:

1) Docker docs. Docker Overview. URL: https://docs.docker.com/engine/docker-overview/

2) Docker docs. Orientation and setup. URL: https://docs.docker.com/get-started/