Docker Assignment 7

1. In Docker, where are docker volumes stored?

Volumes are also stored as part of the host file system, which is managed by Docker. On Linux, volumes are stored in "/var/lib/docker/volume".

To list them:

sudo docker volume ls

2. What is the purpose of the docker info command?

This command displays system wide information regarding the Docker installation. Information displayed includes the kernel version, number of containers and images. The number of images shown is the number of unique images. The same image tagged under different names is counted only once.

3. What are the minimum system requirements for Docker to run?

- Operating Systems-MacOS, Linux, Windows 10 Professional or Enterprise edition.
- CPUs-4 cores
- Memory-8 GB
- BIOS-level hardware virtualization support must be enabled in the BIOS settings
- Hyper-V and Containers Windows features must be enabled.

4. What are the most widely utilized Dockerfile instructions?

- **FROM** – Defines the ubuntu:16.04 base image to use.
- **RUN** – Executes commands in a new layer on the top of the base image.
- **CMD** – CMD allows you to run the commands. There are two ways in which commands are executed either via exec or using shell formats.
- **EXPOSE** – Informs Docker that the container listens on the specified network ports at runtime.
- **WORKDIR** - You can specify your working directory inside the container using the WORKDIR instruction. Any other instruction after that in the dockerfile, will be executed on that particular working directory only.
- **COPY** - This instruction allows you to copy a directory from your local machine to the docker container.
- **ADD** - Similar to COPY instruction, you can use ADD to copy files and folders from your local machine to docker containers. However, ADD also allows you to copy files from a URL as well as a tar file.

5. When using docker-compose, how will you ensure that container 1 runs before container 2? Also, could you provide a sample Yaml example file?

Docker-compose does not wait for any container to be "ready" before going ahead with the next containers. In order to achieve the order of execution, we can use:

The "**depends_on**" which got added in version 2 of docker-compose can be used as shown in a sample docker-compose.yml file below:

Sample yml file:

```
version: "2.4"
services:
 backend:
   build: .
   depends_on:
     - db
 db:
   image: postgres
```

- The **docker-compose up** command starts and runs the services in the dependency order specified. For the above example, the DB container is started before the backend.
- **docker-compose up SERVICE_NAME** by default includes the dependencies associated with the service. In the given example, running **docker-compose up backend** creates and starts DB (dependency of backend).
- Finally, the command **docker-compose stop** also stops the services in the order of the dependency specified. For the given example, the backend service is stopped before the DB service

6. How can I utilise Docker to run several applications in a single environment?

First of all, you need to install docker-compose in your environment.

Create a **docker-compose.yaml** file that defines the services (containers) that make up your application. So they can be run together in an isolated environment. In this compose file, we define all the configurations that need to build and run the services as docker containers.

There are several steps to follow to use docker-compose.

1. Split your app into services

The first thing to do is to think about how you're going to divide the components of your application into different services(containers).
In a simple client-server web application, it could contain three main layers (frontend, backend, and the database). So we can split the app in that way. Likewise, you will have to identify your services of the application, respectively.

2. Pull or build images

For some of your services, you may not need to build from a custom Dockerfile , and a public image on DockerHub will suffice.
For example, if you have a MySQL database in your application, you can pull MySQL image from the hub instead of building it. For others, you will have to create a Dockerfile and build them.

3. Configure environment variables, declare dependencies

Most applications use environment variables for initialization and startup. And also, after we divide the application into services, they have

dependencies on each other. So we need to identify those things before we declare the compose file.

4. Configure networking

Docker containers communicate with each other through their internal network that is created by compose (eg service_name:port). If you want to connect from your host machine, you will have to expose the service to a host port.

5. Set up volumes

In most cases, we would not want our database contents to be lost each time the database service is brought down. A simple way to persist our DB data is to mount a volume.

6. Build & Run

Now, you are set to go and create the compose file and build the images for your services and generate containers from those images.

7. What are the System commands for controlling Docker?

The parent command for docker is

docker The base command for the Docker CLI.

The child commands for docker are:

docker system df-Show docker disk usage

docker system events-Get real time events from the server

docker system info-Display system-wide information

docker system prune-Remove unused data

8. How do you scale your Docker containers, and what command do you use to do so?

The scale command enables you to scale one or more replicated services either up or down to the desired number of replicas. This command cannot be applied on services which are global mode. The command will return immediately, but the actual scaling of the service may take some time. To stop all replicas of a service while keeping the service active in the swarm you can set the scale to 0.

docker service scale SERVICE=REPLICAS

9. What exactly is CNM?

Container Network model(CNM) provides the forwarding rules, network segmentation, and management tools for complex network policies. It formalizes the steps required to enable networking for containers while providing an abstraction that can be used to support multiple network drivers. Docker uses several networking technologies to implement the CNM network drivers including Linux bridges, network namespaces, veth pairs, and iptables.

The CNM is built on three components, sandbox, endpoint, network:

Sandbox

contains the configuration of a container's network stack.

Endpoint

joins a Sandbox to a Network

Network

group of Endpoints that can directly communicate with one other implemented as a Linux bridge, a VLAN, etc.

10. Explain what some of the more complex Docker commands do.

Some advanced commands include:

Docker info- Displays system-wide information regarding the Docker installation

Docker pull -Downloads an image

Docker stats - Provides you with container information

Docker images- Lists downloaded images

11. How do Docker object labels work?

A label is a key-value pair, stored as a string. You can specify multiple labels for an object, but each key-value pair must be unique within an object. If the same key is given multiple values, the most-recently-written value overwrites all previous values.

To demonstrate the labeling on images, I'll create the following simple Dockerfileand build an image out of it:

FROM ubuntu

ARG buildDate

LABEL buildDate=$buildDate

The first line of the file means that we're going to create an image based on the ubuntu.
The second line, starting with ARG, defines an argument that is expected during the build time.
The third line, starting with LABEL, sets a label for this image. The label value will be provided from the command line during the build as the buildDate argument. Its value will be substituted instead of the placeholder.