Experiment-9

Aim: To study and Implement Containerization using Docker.

Software Required: Docker (CLI)

Theory:
Containerization or isolation is the packaging together of software code with all it's necessary components like libraries, frameworks, and other dependencies so that they are isolated in their own 'container'.

This is so that the software or application within the container can be moved and run consistently in any environment and on any infrastructure, independent of that environment or infrastructure's operating system. The container acts as a kind of bubble or a computing environment surrounding the application and keeping it independent of its surroundings. It's basically a fully functional and portable computing environment.

Containers are an alternative to coding on one platform or operating system, which made moving their application difficult since the code might not then be compatible with the new environment. This could result in bugs, errors, and glitches that needed fixing (meaning more time, less productivity, and a lot of frustration).

By packaging up an application in a container that can be moved across platforms and infrastructures, that application can be used wherever you move it because it has everything it needs to run successfully within it.

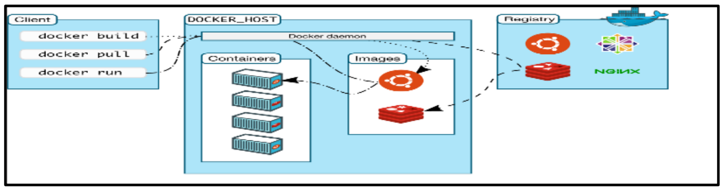
Docker

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allows you to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you do not need to rely on what is currently installed on the host. You can easily share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

Docker uses a client-server architecture. The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers.

The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, which lets you work with applications consisting of a set of containers.



Components of Docker

1. **Docker Daemon**:
   The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.
2. **Docker Client**:
   The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.
3. **Docker Desktop**:
   Docker Desktop is an easy-to-install application for your Mac or Windows environment that enables you to build and share containerized applications and microservices. Docker Desktop includes the Docker daemon (dockerd), the Docker client (docker), Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper.
4. **Docker registries**:
   A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry. When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.
5. **Docker objects**:
   When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

Docker demonstration

Creating a container

1. For creating a docker container, we can use the docker client's *create* option. The *docker create* command creates a writeable container layer over the specified image and prepares it for running the specified command. The container ID is then printed to STDOUT. This is similar to *docker run -d* except the container is never started. You can then use the docker start command to start the container at any point. See examples below.
   Example:

```
$ docker create -t -i fedora bash
6d8af538ec541dd581ebc2a24153a28329acb5268abe5ef868c1f1a26
$ docker start -a -i 6d8af538ec5
bash-4.2#
```

Finding a Docker container

To find a specific docker container, we can use the *docker* client's *ps* command, which will list the container(s). This works somewhat like Linux's *ps* command. This way, we can view all of the containers via their names, IDs or their status, or filter the containers according to our needs. See examples below.

```
// To show only running containers
$ docker ps
// To show all containers
$ docker ps -a
// To show a specific container by its ID
$ docker ps --filter "id=6d8af538ec5"
```

Building a docker container from a Dockerfile

Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using *docker build* users can create an automated build that executes several command-line instructions in succession. The *docker build* command builds Docker images from a Dockerfile and a "context". A build's context is the set of files located in the specified PATH or URL. The build process can refer to any of the files in the

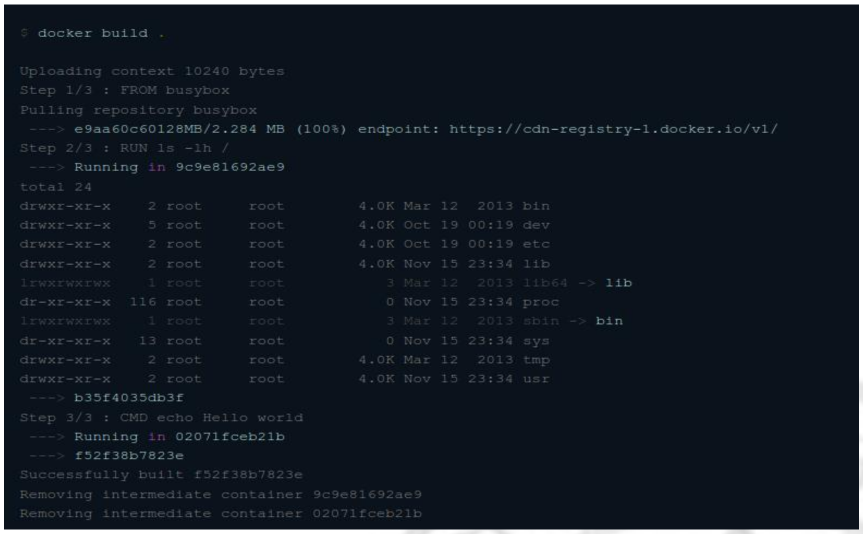
context. For example, your build can use a COPY instruction to reference a file in the context. See examples below.

The file below is a sample Dockerfile, which sets up tools for running *node.js*

```
# syntax=docker/dockerfile:1
FROM node:12-alpine
RUN apk add --no-cache python2 g++ make
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```

We can execute this Dockerfile to run our docker container. We can execute the command below in the same directory as the Dockerfile.

```
$ docker build -t node-js .
```



```
$ docker build .

Uploading context 10240 bytes
Step 1/3 : FROM busybox
Pulling repository busybox
 ---> e9aa60c60128MB/2.284 MB (100%) endpoint: https://cdn-registry-1.docker.io/v1/
Step 2/3 : RUN ls -lh /
 ---> Running in 9c9e81692ae9
total 24
drwxr-xr-x    2 root     root        4.0K Mar 12  2013 bin
drwxr-xr-x    5 root     root        4.0K Oct 19 00:19 dev
drwxr-xr-x    2 root     root        4.0K Oct 19 00:19 etc
drwxr-xr-x    2 root     root        4.0K Nov 15 23:34 lib
lrwxrwxrwx    1 root     root           3 Mar 12  2013 lib64 -> lib
dr-xr-xr-x  116 root     root           0 Nov 15 23:34 proc
lrwxrwxrwx    1 root     root           3 Mar 12  2013 sbin -> bin
dr-xr-xr-x   13 root     root           0 Nov 15 23:34 sys
drwxr-xr-x    2 root     root        4.0K Mar 12  2013 tmp
drwxr-xr-x    2 root     root        4.0K Nov 15 23:34 usr
 ---> b35f4035db3f
Step 3/3 : CMD echo Hello world
 ---> Running in 02071fceb21b
 ---> f52f38b7823e
Successfully built f52f38b7823e
Removing intermediate container 9c9e81692ae9
Removing intermediate container 02071fceb21b
```

Running a Docker container

Once we install a docker container, either by creating it with *docker create* or by using *docker build* with a Dockerfile, we can run the container to actually use the container and the services that the container provides. We can use the *docker* client's *start* command to start a container. Docker allows starting containers by their names or their IDs. See examples below.

```
// Create container

$ docker create --name my_container -t -i fedora bash

// Start container by name

$ docker start my_container

// Start container by ID

$ docker start -a -i 6d8af538ec5
```

Conclusion:-

Sign and Remark:

R1 (3 Marks)	**R2 (2 Marks)**	**R3 (5 Marks)**	**R4 (5 Mark)**	**Total (15 Marks)**	**Signature**