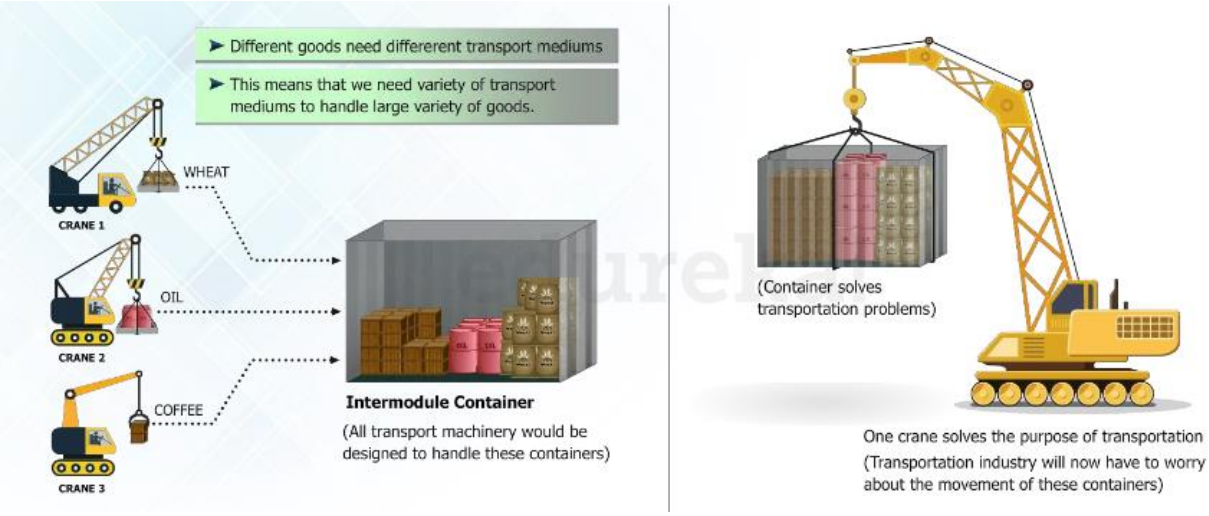
DOCKER

Before diving deep into Dockers, Let's understand

- Why do we need Docker?

Need of Docker:



Current IT industry Problem:

We have a similar issue to the one seen by the transport industry – we have to continually invest substantial manual effort to move code between different environments.

A typical modern system may include java script framework, NoSQL database, message queues, REST APIs and back-ends all written in a variety of programming language.

This stack has to run partly or completely on top of variety of hardware - from the developer's laptop and the in-house testing cluster to the production cloud provider.

Solution:

Much as the intermodal containers simplified the transportation of goods, Docker containers simplify the transportation of software applications. Developers can concentrate on building the application on building the application and shipping it through testing and production without worrying about differences in environment and dependencies.

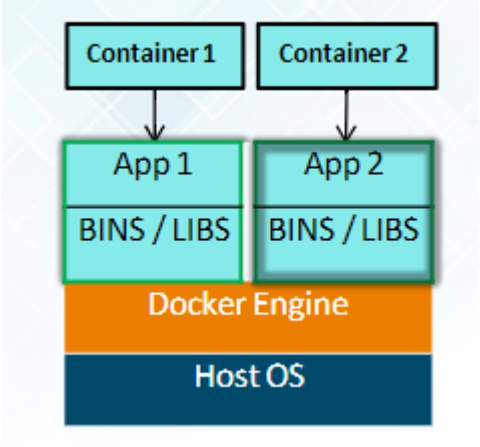
Operations can focus on the core issues of running containers, such as allocating resources, starting and stopping containers, and migrating them between servers.

Till now we have seen why do we need Docker ? Now let's understand:

- What is Docker ?
- The differences betwwen Containers nd VMs

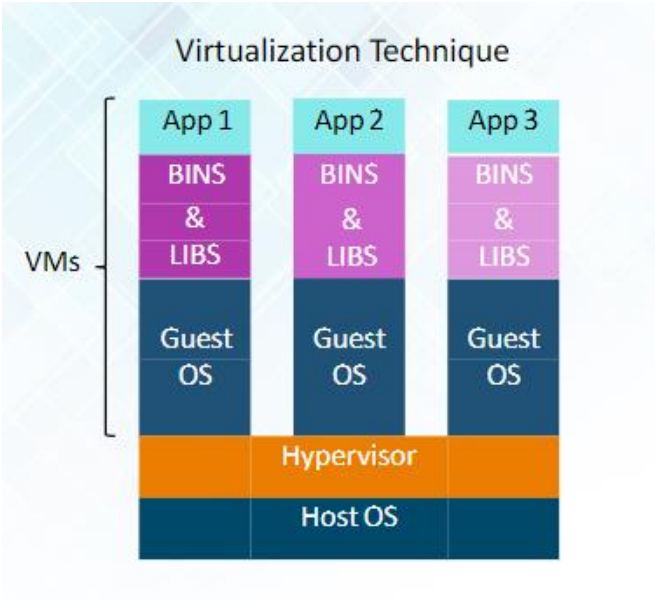
What is Docker?

Docker is a Containerization platform which package your application and all its dependencies together in the form of Containers so as to ensure that your application works seamlessly in any environment, be it Development or Test or Production.



The differences betwwen Containers and VMs.

A. Virtualization, Adopted by VMs



Advantages

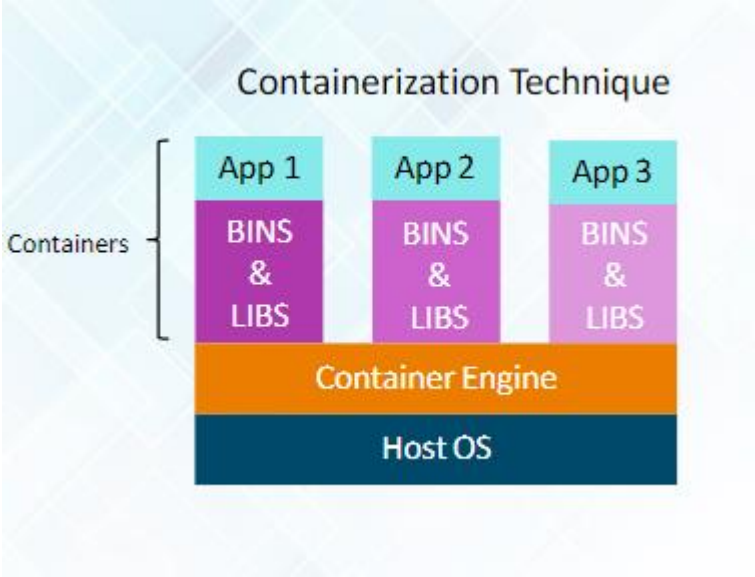
- Multiple OS in same machine
- Easy to maintenance and recovery
- Lower total cost of ownership

Disadvantages

Multiple VMs lead to unstable performance Hypervisors are not as efficient as host OS

Long boot-up process (approx. 1 min)

B. Containerization

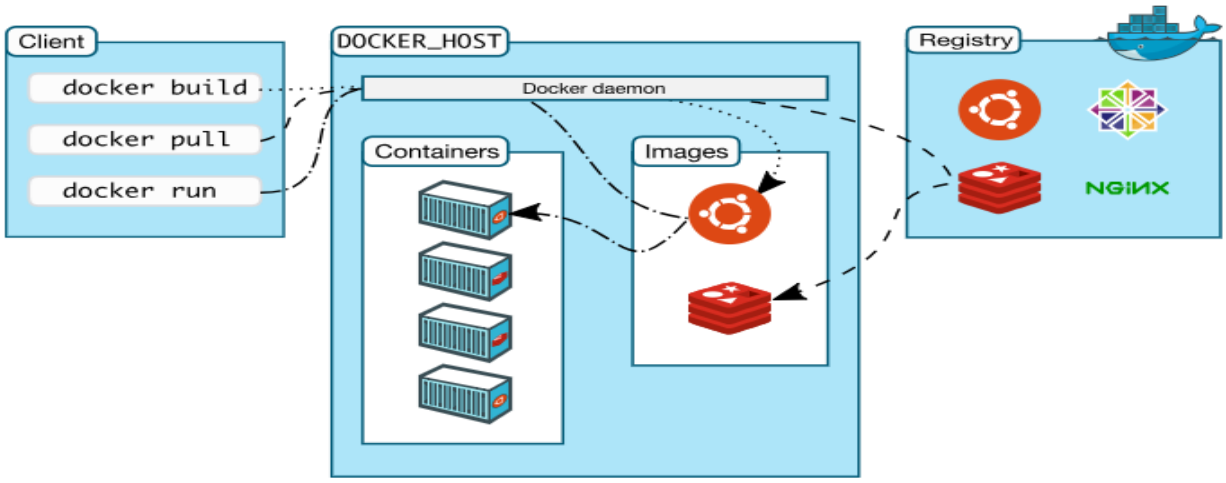


Advantages over Virtualization:

1. Containers on same OS kernel are lighter and smaller
2. Better resourc utilization compared to VMs
3. Short boot-up process (1/20th of a sec.)

Docker Architecture:

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.



The 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.

The 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.

Docker registries

A Docker registry stores Docker images. Docker Hub and Docker Cloud are public registries 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. If you use Docker Datacenter (DDC), it includes Docker Trusted Registry (DTR).

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.

Docker store allows you to buy and sell Docker images or distribute them for free. For instance, you can buy a Docker image containing an application or service from a software vendor and use the image to deploy the application into your testing, staging, and production environments. You can upgrade the application by pulling the new version of the image and redeploying the containers.

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.

IMAGES

An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a Dockerfile with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

CONTAINERS

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

USECAES: Docker installation and Docker hub setup.

Step1: Install Docker

The Docker installation package available in the official Ubuntu 16.04 repository may not be the latest version. To get the latest and greatest version, install Docker from the official Docker repository.

First, add the GPG key for the official Docker repository to the system:

```
$ curl -fssl https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Add the Docker repository to APT sources:

```
$ sudo add-apt-repository "deb
[arch=amd64]https://download.docker.com/linux/ubuntu$(lsb_release -cs) stable"
```

Next, update the package database with the Docker packages from the newly added repo:

```
$ sudo apt-get update
```

Make sure you are about to install from the Docker repo instead of the default Ubuntu 16.04 repo:

```
$ apt-cache policy docker-ce
```

You should see output similar to the follow:

Output of apt-cache policy docker-ce

```
docker-ce:
  Installed: (none)
  Candidate: 17.03.1~ce-0~ubuntu-xenial
  Version table:
     17.03.1~ce-0~ubuntu-xenial 500
        500 https://download.docker.com/linux/ubuntu xenial/stable amd64 Packages
     17.03.0~ce-0~ubuntu-xenial 500
        500 https://download.docker.com/linux/ubuntu xenial/stable amd64 Packages
```

Installation is from the Docker repository for Ubuntu 16.04. The docker-ce version number might be different.

Finally, install Docker:

```
$ sudo apt-get install -y docker-ce
```

Docker should now be installed, the daemon started, and the process enabled to start on boot. Check that it's running:

```
$ sudo systemctl status docker
```

The output should be similar to the following, showing that the service is active and running:

```
Output
docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2016-05-01 06:53:52 CDT; 1 weeks 3 days ago
     Docs: https://docs.docker.com
 Main PID: 749 (docker)
```

Installing Docker now gives you not just the Docker service (daemon) but also the docker command line utility, or the Docker client.

Step2: To create an account on Docker Hub, register at Docker Hub. Afterwards, to push your image, first log into Docker Hub. You'll be prompted to authenticate:

```
$ docker login -u docker-registry-username
```

If you specified the correct password, authentication should succeed. Then you may push your own image using:

```
$ docker push docker-registry-username/docker-image-name
```

USECASE: How bring up static web server with Docker using Nginx image.

NGINX is a free, open-source, high-performance HTTP server and reverse proxy, as well as an IMAP/POP3 proxy server. NGINX is known for its high performance, stability, rich feature set, simple configuration, and low resource consumption.
For more information visit https://www.nginx.com/resources/wiki/

Step1: Pull an image from Docker-hub.

```
$ docker pull nginx:alpine
```

```
root@linux2:~# docker pull nginx:alpine
alpine: Pulling from library/nginx
911c6d0c7995: Pull complete
a3b372682d36: Pull complete
e910d9195403: Pull complete
05b1fd5fddf1: Pull complete
Digest: sha256:2c4269d573d9fc6e9e95d5e8f3de2dd0b07c19912551f25e848415b5dd783acf
Status: Downloaded newer image for nginx:alpine
root@linux2:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx               alpine              336262580e12        2 days ago          18.6MB
root@linux2:~#
```

Step2: Run this image in a container and expose port 80.

```
$ docker run –d –p 8060:80 nginx
```

```
root@linux2:~# docker run -d -p 8060:80 nginx
04685ceafb8ddb46ee105b196bd92383e060062ba2e3e18e677589cb7cd1eb87
root@linux2:~#
```

Step3: Enter inside the container and you will see a index.html file.

$ docker exec –it <container_name> \sh

```
root@linux2:~# docker exec -it laughing_ritchie \sh
# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
# cd usr/share/nginx/html
# ls
50x.html  index.html
# cat index.html
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
#
```

Next, come out of container. Press ctrl+p, ctrl+q. Now, open your web browser and open a url http://<ipaddress>:8060
You will get a web page as given below.

How to commit and save changes made to container and use as docker image

Step1: Create a container from ubuntu:latest image and a bash terminal.

```
root@linux2:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ngnix_application   latest              dae9bb4fbd7a        25 minutes ago      18.6MB
ubuntu              base                d16fb992d911        6 hours ago         139MB
static_app          latest              845dc65933e5        11 hours ago        18.6MB
ubuntu              latest              cd6d8154f1e1        43 hours ago        84.1MB
nginx               alpine              336262580e12        2 days ago          18.6MB
root@linux2:~#
```

$ docker run –it ubuntu:latest /bin/bash

```
root@linux2:~# docker run -it ubuntu:latest /bin/bash
root@3fbfd1d1772a:/# java
```

Step2: Inside ubuntu container type java. You will come to know that java is no yet configred in this container.

```
root@3fbfd1d1772a:/# java
bash: java: command not found
root@3fbfd1d1772a:/# javac
bash: javac: command not found
root@3fbfd1d1772a:/#
```

Step3: Install java in this container. To install java execute these coomands.

$ apt-get update && apt-get upgrade

```
root@3fbfd1d1772a:/# apt-get update && apt-get upgrade
Get:1 http://security.ubuntu.com/ubuntu bionic-security InRelease [83.2 kB]
Get:2 http://archive.ubuntu.com/ubuntu bionic InRelease [242 kB]
Get:3 http://security.ubuntu.com/ubuntu bionic-security/universe Sources [17.4 kB]
Get:4 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:5 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages [203 kB]
```

$ apt-get install default-jdk

```
root@3fbfd1d1772a:/# apt-get install default-jdk
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  adwaita-icon-theme at-spi2-core ca-certificates ca-certificates-java dbus dconf-gsettings-backend dconf-service default-jdk-headless de
  default-jre-headless fontconfig fontconfig-config fonts-dejavu-core fonts-dejavu-extra glib-networking glib-networking-common glib-netw
  gsettings-desktop-schemas gtk-update-icon-cache hicolor-icon-theme humanity-icon-theme java-common krb5-locales libapparmor1 libasound2
```

Now, check if java is installed or not.

```
root@3fbfdld1772a:/# java
Usage: java [options] <mainclass> [args...]
           (to execute a class)
   or  java [options] -jar <jarfile> [args...]
           (to execute a jar file)
   or  java [options] -m <module>[/<mainclass>] [args...]
       java [options] --module <module>[/<mainclass>] [args...]
           (to execute the main class in a module)

 Arguments following the main class, -jar <jarfile>, -m or --module
 <module>/<mainclass> are passed as the arguments to main class.

 where options include:
```

So, java is installed in this contaier.

Step4: Now, come out of container and save this container as new image.

To come out of a container type ctrl+p , ctrl+q.

And to save a container as a new image, you need to commit this container as given below:

$ docker commit <conainer_id> <new_image_name>

```
root@linux2:~# docker ps
CONTAINER ID        IMAGE                   COMMAND                  CREATED             STATUS              PORTS
3fbfdld1772a        ubuntu:latest           "/bin/bash"              12 minutes ago      Up 12 minutes
40c776d3d917        ngnix_application:latest   "nginx -g 'daemon of…"   33 minutes ago      Up 33 minutes       0.0.0.0:8095->80/tcp
root@linux2:~# docker commit 3fbfdld1772a jdk:test
sha256:456aa589e8024cf7f0966dd92bd5d26a631336edf8f567e885dc8e33406fc8b6
root@linux2:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
jdk                 test                456aa589e802        9 seconds ago       632MB
ngnix_application   latest              dae9bb4fbd7a        42 minutes ago      18.6MB
ubuntu              base                d16fb992d911        6 hours ago         139MB
static_app          latest              845dc65933e5        11 hours ago        18.6MB
ubuntu              latest              cd6d8154fle1        44 hours ago        84.1MB
```

USECASE: How to mount local web-app code to Docker container and setup Nginx webserver.

Step1: Create a index.html file in a source folder '/home/docker_example/html_pages'which we will use as a webpage for nginx web serve.

Step2: Pull an image from docker hub or if you already have that then skip this step.

$ docker pull nginx:alpine

_

```
root@linux2:~# docker pull nginx:alpine
alpine: Pulling from library/nginx
911c6d0c7995: Pull complete
a3b372682d36: Pull complete
e910d9195403: Pull complete
05b1fd5fddf1: Pull complete
Digest: sha256:2c4269d573d9fc6e9e95d5e8f3de2dd0b07c19912551f25e848415b5dd783acf
Status: Downloaded newer image for nginx:alpine
root@linux2:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx               alpine              336262580e12        2 days ago          18.6MB
root@linux2:~#
```

Step3: Start a container with bind mount.

Docker has two options for containers to store files in the host machine, so that the files are persisted even after the container stops: volumes, and bind mounts. If you're running Docker on Linux you can also use a tmpfs mount.

- Volumes are stored in a part of the host filesystem which is managed by Docker(/var/lib/docker/volumes/ on Linux). Non-Docker processes should not modify this part of the filesystem. Volumes are the best way to persist data in Docker.
- Bind mounts may be stored anywhere on the host system. They may even be important system files or directories. Non-Docker processes on the Docker host or a Docker container can modify them at any time.
- tmpfs mounts are stored in the host system's memory only, and are never written to the host system's filesystem.

Note: In this step we will going to use bind mounts.

Bind mounts: Available since the early days of Docker. Bind mounts have limited functionality compared to volumes. When you use a bind mount, a file or directory on the host machine is mounted into a container. The file or directory is referenced by its full path on the host machine. The file or directory does not need to exist on the Docker host already. It is created on demand if it does not yet exist. Bind mounts are very performant, but they rely on the host machine's filesystem having a specific directory structure available. If you are developing new Docker applications, consider using named volumes instead. You can't use Docker CLI commands to directly manage bind mounts.

```
$ docker run -d –mount
type=bind,source=/home/docker_example/html_pages/,target=/usr/share/nginx/html/ -p
8095:80 nginx
```

```
root@linux2:~# docker run -d --mount type=bind,source=/home/docker_example/html_pages/,target=/usr/share/nginx/html/ -p 8095:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
802b00ed6f79: Pull complete
e9d0e0ea682b: Pull complete
d8b7092b9221: Pull complete
Digest: sha256:24a0c4b4a4c0eb97a1aabb8e29f18e917d05abfe1b7a7c07857230879ce7d3d3
Status: Downloaded newer image for nginx:latest
9349la6f08cf81211ffdd7cc6101d4712e0020b716792abda2c5979afac8eeba
root@linux2:~# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
9349la6f08cf        nginx               "nginx -g 'daemon of…"   22 seconds ago      Up 10 seconds       0.0.0.0:8095->80/tcp     pedantic_swanson
root@linux2:~#
```

Next, open your browser and open a url http://<ipaddress>/8095. You will get a web page.

USECASE: How create a custom Docker image using Dockerfile

In this usecase we will use nginx web server and we are going to host a static web page inside this image. We will create a custom image for that using Docker file.

Dockerfile: it defines what goes on in the environment inside your container. Access to resources like networking interfaces and disk drives is virtualized inside this environment, which is isolated from the rest of your system, so you need to map ports to the outside world, and be specific about what files you want to "copy in" to that environment. However, after doing that, you can expect that the build of your app defined in this Dockerfile behaves exactly the same wherever it runs.

Step 1: Create a simple html file named index.html in a folder.

```
root@linux2:~/SEC_scenario# cat index.html
<!DOCTYPE html>
<html>
<head>
<style>
body {
    background-color: lightblue;
}

h1 {
    color: white;
    text-align: center;
}

p {
    font-family: verdana;
    font-size: 20px;
}
</style>
</head>
<body>

<h1>Hello World</h1>
<p>Here I'm bringing up a static web server using nginx </p>

</body>
</html>

root@linux2:~/SEC_scenario#
```

Step 2: Create a Docker file.

```
root@linux2:~/SEC_scenario# cat Dockerfile
FROM nginx:alpine
COPY .  /usr/share/nginx/html
EXPOSE 80
root@linux2:~/SEC_scenario#
```

Step 3: Build this Dockerfile

```
$ docker build –t <custom_image_name> .
```

Dot(.) is to tell docker CLI that Docker file is in same dir.

```
root@linux2:~/SEC_scenario# ls
Dockerfile  index.html
root@linux2:~/SEC_scenario# docker build -t ngnix_application .
Sending build context to Docker daemon  3.072kB
Step 1/3 : FROM nginx:alpine
 ---> 336262580e12
Step 2/3 : COPY .  /usr/share/nginx/html
 ---> 71386f24dd3a
Step 3/3 : EXPOSE 80
 ---> Running in cda25f1bdc28
Removing intermediate container cda25f1bdc28
 ---> dae9bb4fbd7a
Successfully built dae9bb4fbd7a
Successfully tagged ngnix_application:latest
root@linux2:~/SEC_scenario#
```

Now, you will find your image ngnix_application:latest. This is your custom image that you have just created from Dockerfile.

USECASE: How to build a custom web-app image and link to standard mysql db container using docker-compose.