Name- Ayan Mondal

Employee ID-815954

**Docker Notes:-**

Docker is a platform for packaging, deploying, and running applications. Docker applications run in containers that can be used on any system: a developer’s laptop, systems on premises, or in the cloud.

[Containerization](https://searchitoperations.techtarget.com/definition/application-containerization-app-containerization) is a technology that’s been around for a long time, but it’s seen new life with [Docker](https://stackify.com/complete-docker-toolkit/). It packages applications as images that contain everything needed to run them: code, runtime environment, libraries, and configuration. Images run in containers, which are discrete processes that take up only as many resources as any other executable.

It’s important to note that Docker containers don’t run in their own virtual machines, but share a Linux kernel. Compared to virtual machines, containers use less memory and less CPU.

However, a Linux runtime is required for Docker. Implementations on non-Linux platforms such as macOS and Windows 10 use a single Linux virtual machine. The containers share this system.

[Containerization](https://stackify.com/microservices-containers/) has enjoyed widespread adoption because of its

* Consistent test environment for development and QA.
* Cross-platform packages called images.
* Isolation and encapsulation of application dependencies.
* Ability to scale efficiently, easily, and in real time.
* Enhances efficiency via easy reuse of images.

**docker run hello-world** does exactly what it sounds like. It runs an image named “hello-world.”

Docker looks for this image on our local system. When it can’t find the image, Docker downloads it from [Docker Hub](https://hub.docker.com/) for us.

Hello-world displays a message telling us everything’s working. Then it spells out the process for us before recommending some additional steps

### Reuse a container

This time, we used **docker start –attach <container name>** instead of **docker** **run**.We use the **start** command, and rather than naming the image, we specify the name of a container that’s already loaded. The **–attach** tells Docker to connect to the container output so we can see the results.

We stop containers with **docker stop <container name>** and remove them with **docker rm <container name>**. We’ll take a look at that below when we work with applications designed to keep running in the background.

If we check **docker ps** again, we still see two containers.

We can stop it with **docker stop**…

$ docker stop compassionate\_ritchie

…and remove the container with **docker rm.**

$ docker rm compassionate\_ritchie

After running these two commands, the container is gone:

### Create a Docker image

Now let’s build on this example to create an image of our own. We’ll package the Nginx image with our **html** file.

Images are created with a [Dockerfile,](https://docs.docker.com/engine/reference/builder/) which lists the components and commands that make up an image.

In **my-nginx**, create a Dockerfile:

FROM nginx

COPY html /usr/share/nginx/html

This Dockerfile contains two instructions:

1. First, create this image from an existing image, which is named **nginx.** The **FROM** instruction is a requirement for all Dockerfiles and establishes the base image. Subsequent instructions are executed on the base image.
2. The second instruction, **COPY,** tells Docker to copy our file tree into the base image, overriding the contents of **/usr/share/nginx/html** in the base image.

Next, build the image:

$ docker build -t mynginx .

Sending build context to Docker daemon 3.584kB

Step 1/2 : FROM nginx

---> b175e7467d66

Step 2/2 : COPY html /usr/share/nginx/html

---> Using cache

---> a8b02c2e09a4

Successfully built a8b02c2e09a4

Successfully tagged mynginx:latest

We passed two arguments to **build**:

* **-t mynginx** gave Docker a tag for the image. Since we only supplied a name, we can see that Docker tagged this build as the latest in the last line of the build output. We’ll look more closely at tagging below.
* The final argument, **dot (**or “.”), told Docker to look for the Dockerfile in the current working directory.

The build output shows Docker using the **nginx** image and copying the contents of **html** into the new image.

### Run a custom image

Next, we run our new image:

$ docker run --name foo -d -p 8080:80 mynginx

Let’s break that command down.

* **–name foo** gives the container a name, rather than one of the randomly assigned names we’ve seen so far.
* **-d** detaches from the container, running it in the background, as we did in our previous run of Nginx.
* **-p 8080:80** maps network ports, as we did with the first example.
* Finally, the image name is always last.

Now point your browser at [**http://127.0.0.1:8080**](http://127.0.0.1:8080/) and you can see the test web page again.