Vishnutejakankanala / Docker-Zero-to-Hero

Code Pull requests Actions Projects Wiki Security Insights

Docker-Zero-to-Hero / README.md

iam-veeramalla Update README.md 2 weeks ago

362 lines (213 loc) · 14.4 KB

Preview Code Blame Raw

Repo to learn Docker with examples. Contributions are most welcome.

If you found this repo useful, give it a STAR 🌠

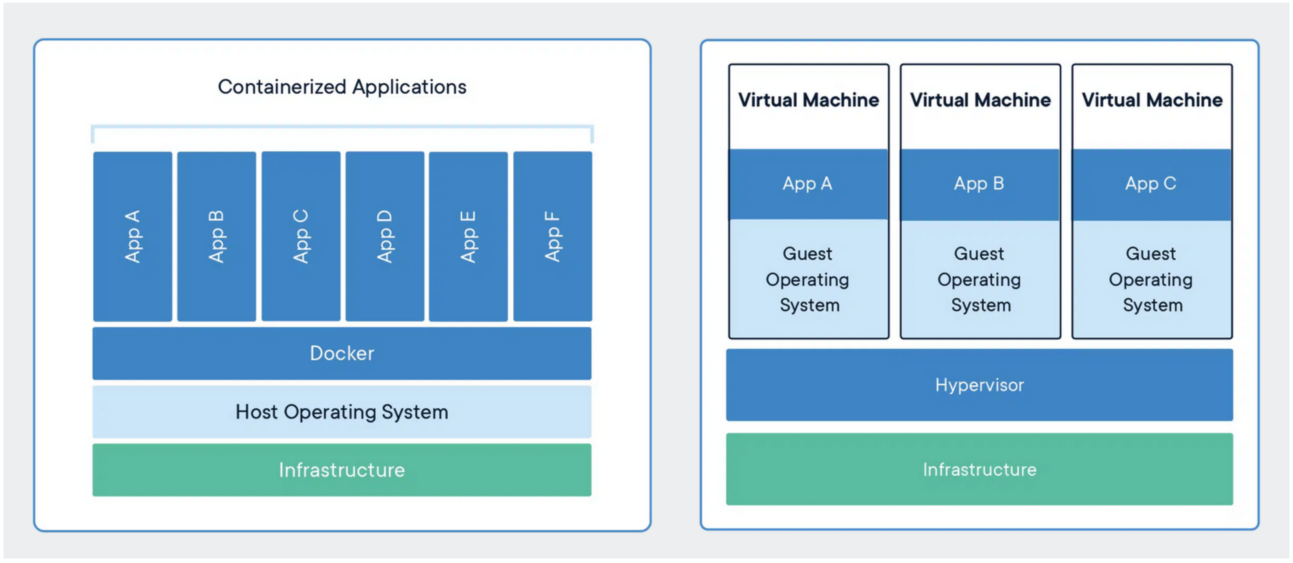
You can watch the video version of this repo on my youtube playlist. -> https://www.youtube.com/watch?v=7JZP345yVjw&list=PLdpzxOOAlwvLjb0vTD9BXLOwwLD_GWCmC

What is a container ?

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Ok, let me make it easy !!!

A container is a bundle of Application, Application libraries required to run your application and the minimum system dependencies.



Containers vs Virtual Machine

Containers and virtual machines are both technologies used to isolate applications and their dependencies, but they have some key differences:

```
1. Resource Utilization: Containers share the host operating system kernel,
making them lighter and faster than VMs. VMs have a full-fledged OS and
hypervisor, making them more resource-intensive.

2. Portability: Containers are designed to be portable and can run on any
system with a compatible host operating system. VMs are less portable as
they need a compatible hypervisor to run.

3. Security: VMs provide a higher level of security as each VM has its own
operating system and can be isolated from the host and other VMs.
Containers provide less isolation, as they share the host operating system.
```

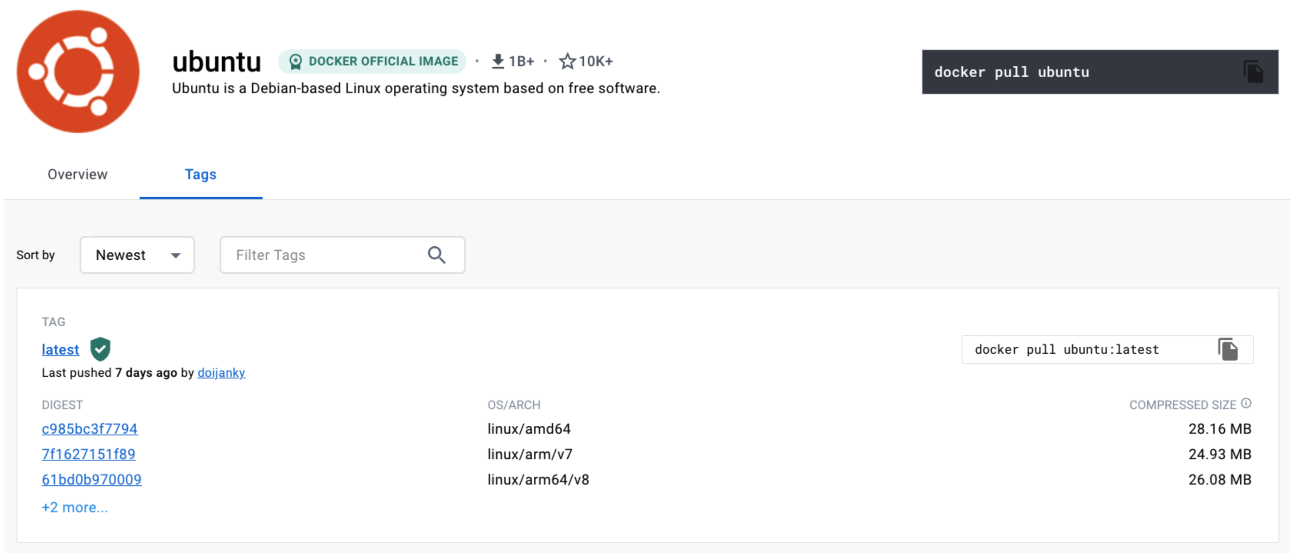
4. Management: Managing containers is typically easier than managing VMs, as containers are designed to be lightweight and fast-moving.

Why are containers light weight ?

Containers are lightweight because they use a technology called containerization, which allows them to share the host operating system's kernel and libraries, while still providing isolation for the application and its dependencies. This results in a smaller footprint compared to traditional virtual machines, as the containers do not need to include a full operating system. Additionally, Docker containers are designed to be minimal, only including what is necessary for the application to run, further reducing their size.

Let's try to understand this with an example:

Below is the screenshot of official ubuntu base image which you can use for your container. It's just ~ 22 MB, isn't it very small ? on a contrary if you look at official ubuntu VM image it will be close to ~ 2.3 GB. So the container base image is almost 100 times less than VM image.



Ubuntu is a Debian-based Linux operating system based on free software.

```
docker pull ubuntu
```

TAG
latest
Last pushed 7 days ago by doijanky

```
docker pull ubuntu:latest
```

DIGEST	OS/ARCH	COMPRESSED SIZE
c985bc3f7794	linux/amd64	28.16 MB
7f1627151f89	linux/arm/v7	24.93 MB
61bd0b970009	linux/arm64/v8	26.08 MB
+2 more...		

To provide a better picture of files and folders that containers base images have and files and folders that containers use from host operating system (not 100 percent accurate -> varies from base image to base image). Refer below.

Files and Folders in containers base images

```
    /bin: contains binary executable files, such as the ls, cp, and ps
  commands.

    /sbin: contains system binary executable files, such as the init and
  shutdown commands.

    /etc: contains configuration files for various system services.
    /lib: contains library files that are used by the binary executables.

    /usr: contains user-related files and utilities, such as applications,
libraries, and documentation.

    /var: contains variable data, such as log files, spool files, and
temporary files.

    /root: is the home directory of the root user.
```

Files and Folders that containers use from host operating system

```
    The host's file system: Docker containers can access the host file
system using bind mounts, which allow the container to read and write files
in the host file system.

    Networking stack: The host's networking stack is used to provide
network connectivity to the container. Docker containers can be connected
to the host's network directly or through a virtual network.

    System calls: The host's kernel handles system calls from the
container, which is how the container accesses the host's resources, such
as CPU, memory, and I/O.

    Namespaces: Docker containers use Linux namespaces to create isolated
environments for the container's processes. Namespaces provide isolation
for resources such as the file system, process ID, and network.

    Control groups (cgroups): Docker containers use cgroups to limit and
control the amount of resources, such as CPU, memory, and I/O, that a
container can access.
```

It's important to note that while a container uses resources from the host operating system, it is still isolated from the host and other containers, so changes to the container do not affect the host or other containers.

Note: There are multiple ways to reduce your VM image size as well, but I am just talking about the default for easy comparision and understanding.

so, in a nutshell, container base images are typically smaller compared to VM images because they are designed to be minimalist and only contain the necessary components for running a specific application or service. VMs, on the other hand, emulate an entire operating system, including all its libraries, utilities, and system files, resulting in a much larger size.

I hope it is now very clear why containers are light weight in nature.

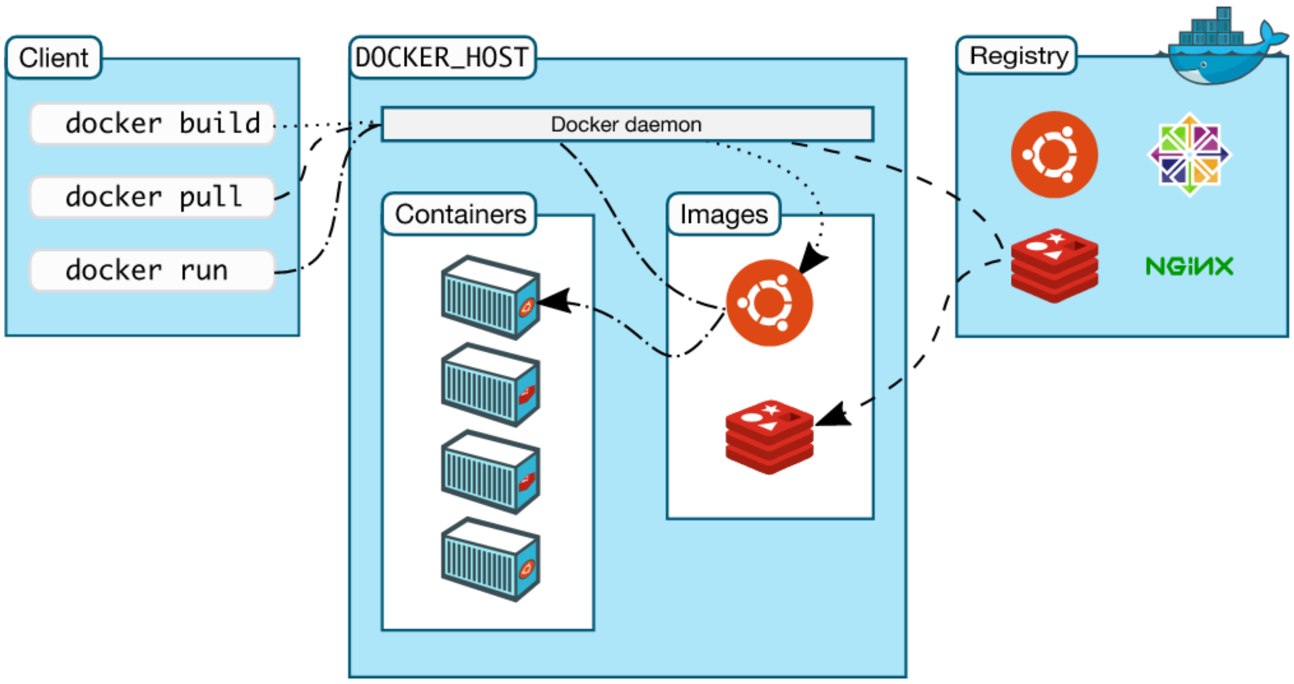
Docker

What is Docker ?

Docker is a containerization platform that provides easy way to containerize your applications, which means, using Docker you can build container images, run the images to create containers and also push these containers to container regestries such as DockerHub, Quay.io and so on.

In simple words, you can understand as `containerization is a concept or technology` and `Docker Implements Containerization`.

Docker Architecture ?



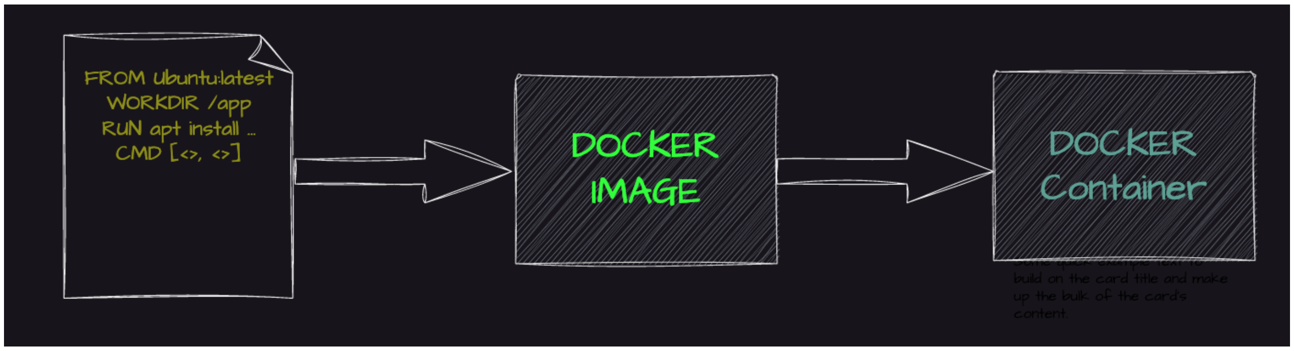
The above picture, clearly indicates that Docker Deamon is brain of Docker. If Docker Deamon is killed, stops working for some reasons, Docker is brain dead :p (sarcasm intended).

Docker LifeCycle

We can use the above Image as reference to understand the lifecycle of Docker.

There are three important things,

1. docker build -> builds docker images from Dockerfile
2. docker run -> runs container from docker images
3. docker push -> push the container image to public/private regestries to share the docker images.



Understanding the terminology (Inspired from Docker Docs)

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.

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 Desktop

Docker Desktop is an easy-to-install application for your Mac, Windows or Linux 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. For more information, see Docker Desktop.

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. 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.

Dockerfile

Dockerfile is a file where you provide the steps to build your Docker Image.

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.

INSTALL DOCKER

A very detailed instructions to install Docker are provide in the below link

https://docs.docker.com/get-docker/

For Demo,

You can create an Ubuntu EC2 Instance on AWS and run the below commands to install docker.

```
sudo apt update
sudo apt install docker.io -y
```

Start Docker and Grant Access

A very common mistake that many beginners do is, After they install docker using the sudo access, they miss the step to Start the Docker daemon and grant acess to the user they want to use to interact with docker and run docker commands.

Always ensure the docker daemon is up and running.

A easy way to verify your Docker installation is by running the below command

```
docker run hello-world
```

If the output says:

```
docker: Got permission denied while trying to connect to the Docker daemon
socket at unix:///var/run/docker.sock: Post
"http://%2Fvar%2Frun%2Fdocker.sock/v1.24/containers/create": dial unix
/var/run/docker.sock: connect: permission denied.
See 'docker run --help'.
```

This can mean two things,

1. Docker deamon is not running.
2. Your user does not have access to run docker commands.

Start Docker daemon

You use the below command to verify if the docker daemon is actually started and Active

```
sudo systemctl status docker
```

If you notice that the docker daemon is not running, you can start the daemon using the below command

```
sudo systemctl start docker
```

Grant Access to your user to run docker commands

To grant access to your user to run the docker command, you should add the user to the Docker Linux group. Docker group is create by default when docker is installed.

```
sudo usermod -aG docker ubuntu
```

In the above command `ubuntu` is the name of the user, you can change the username appropriately.

NOTE: : You need to logout and login back for the changes to be reflected.

Docker is Installed, up and running 🥳 🥳

Use the same command again, to verify that docker is up and running.

```
docker run hello-world
```

Output should look like:

```
....
....
Hello from Docker!
This message shows that your installation appears to be working correctly.
...
...
```

Great Job, Now start with the examples folder to write your first Dockerfile and move to the next examples. Happy Learning :)

Clone this repository and move to example folder

```
git clone https://github.com/iam-veeramalla/Docker-Zero-to-Hero
cd  examples
```

Login to Docker [Create an account with https://hub.docker.com/]

```
docker login
```

```
Login with your Docker ID to push and pull images from Docker Hub. If you
don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: abhishekf5
Password:
WARNING! Your password will be stored unencrypted in
/home/ubuntu/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-
store

Login Succeeded
```

Build your first Docker Image

You need to change the username accoringly in the below command

```
docker build -t abhishekf5/my-first-docker-image:latest .
```

Output of the above command

```
    Sending build context to Docker daemon  992.8kB
    Step 1/6 : FROM ubuntu:latest
    latest: Pulling from library/ubuntu
    677076032cca: Pull complete
    Digest:
sha256:9a0bdde4188b896a372804be2384015e90e3f84906b750c1a53539b585fbbe7f
    Status: Downloaded newer image for ubuntu:latest
     ---> 58db3edaf2be
    Step 2/6 : WORKDIR /app
     ---> Running in 630f5e4db7d3
    Removing intermediate container 630f5e4db7d3
     ---> 6b1d9f654263
    Step 3/6 : COPY . /app
     ---> 984edffabc23
    Step 4/6 : RUN apt-get update && apt-get install -y python3 python3-pip
     ---> Running in a558acdc9b03
    Step 5/6 : ENV NAME World
     ---> Running in 733207001f2e
    Removing intermediate container 733207001f2e
     ---> 94128cf6be21
    Step 6/6 : CMD ["python3", "app.py"]
     ---> Running in 5d60ad3a59ff
    Removing intermediate container 5d60ad3a59ff
     ---> 960d37536dcd
    Successfully built 960d37536dcd
    Successfully tagged abhishekf5/my-first-docker-image:latest
```

Verify Docker Image is created

```
docker images
```

Output

```
REPOSITORY                         TAG       IMAGE ID       CREATED          
SIZE
abhishekf5/my-first-docker-image   latest    960d37536dcd   26 seconds ago   
467MB
ubuntu                             latest    58db3edaf2be   13 days ago      
77.8MB
hello-world                        latest    feb5d9fea6a5   16 months ago    
13.3kB
```

Run your First Docker Container

```
docker run -it abhishekf5/my-first-docker-image
```

Output

```
Hello World
```

Push the Image to DockerHub and share it with the world

```
docker push abhishekf5/my-first-docker-image
```

Output

```
Using default tag: latest
The push refers to repository [docker.io/abhishekf5/my-first-docker-image]
896818320e80: Pushed
b8088c305a52: Pushed
69dd4ccec1a0: Pushed
c5ff2d88f679: Mounted from library/ubuntu
```

```
latest: digest:
sha256:6e49841ad9e720a7baedcd41f9b666fcd7b583151d0763fe78101bb8221b1d88
size: 1157
```

You must be feeling like a champ already