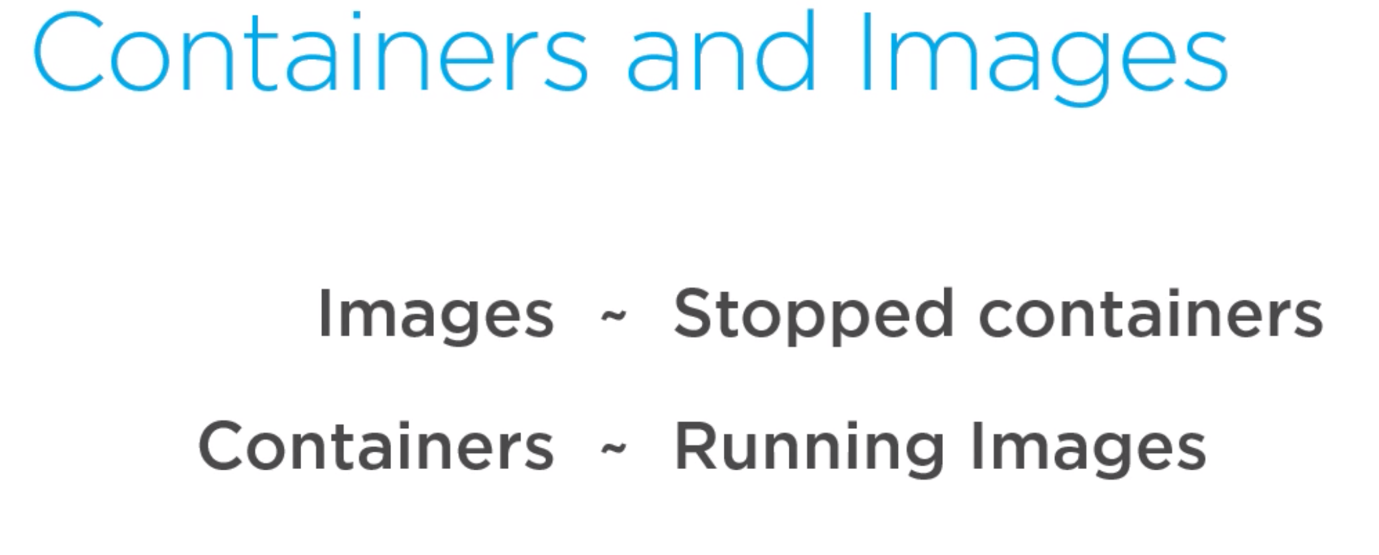
Docker

A close up of text on a white background

Description automatically generated

Diff. between Containers and Images.



About images, containers, and storage drivers

To use storage drivers effectively, you must understand how Docker builds and stores images. Then, you need an understanding of how these images are used by containers. Finally, you’ll need a short introduction to the technologies that enable both images and container operations.

Understanding how Docker manages the data within your images and containers will help you understand the best way to design your containers and Dockerize your applications, and avoid performance problems along the way.

Images and layers

A Docker image is built up from a series of layers. Each layer represents an instruction in the image’s Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:

FROM ubuntu:15.04

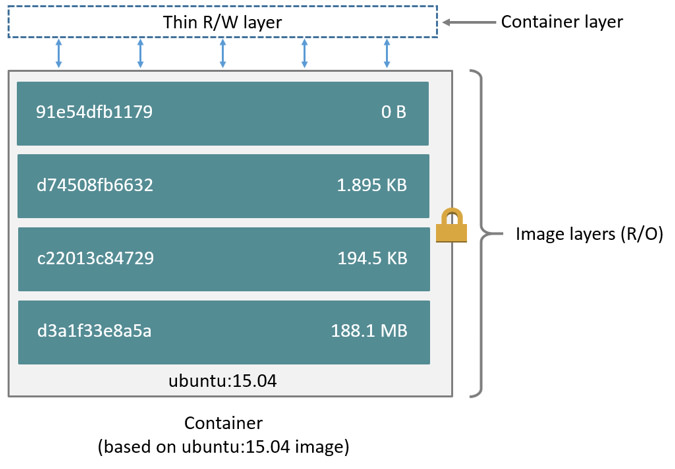
COPY . /app

RUN make /app

CMD python /app/app.py

This Dockerfile contains four commands, each of which creates a layer. The FROM statement starts out by creating a layer from the ubuntu:15.04 image. The COPY command adds some files from your Docker client’s current directory. The RUN command builds your application using the make command. Finally, the last layer specifies what command to run within the container.

Each layer is only a set of differences from the layer before it. The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the “container layer”. All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on the Ubuntu 15.04 image.



A *storage driver* handles the details about the way these layers interact with each other. Different storage drivers are available, which have advantages and disadvantages in different situations.

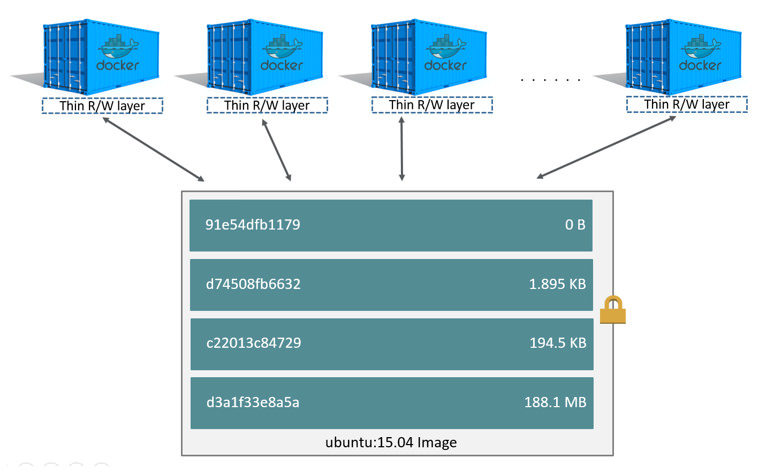
Let’s see an example of having a docker file with multiple layers stacked up to host an asp.net core web api built on a mac machine.

FROM mcr.microsoft.com/dotnet/core/aspnet:2.1-stretch-slim AS base  
WORKDIR /app  
EXPOSE 80  
EXPOSE 443  
  
FROM mcr.microsoft.com/dotnet/core/sdk:2.1-stretch AS build  
WORKDIR /src  
COPY FirstAPI/FirstAPI.csproj FirstAPI/  
RUN dotnet restore "FirstAPI/FirstAPI.csproj"  
COPY . .  
WORKDIR "/src/FirstAPI"  
RUN dotnet build "FirstAPI.csproj" -c Release -o /app  
  
FROM build AS publish  
RUN dotnet publish "FirstAPI.csproj" -c Release -o /app  
  
FROM base AS final  
WORKDIR /app  
COPY --from=publish /app .  
ENTRYPOINT ["dotnet", "FirstAPI.dll"]

Container and layers

The major difference between a container and an image is the top writable layer. All writes to the container that add new or modify existing data are stored in this writable layer. When the container is deleted, the writable layer is also deleted. The underlying image remains unchanged.

Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.



Docker Swarm: A collection of docker engines grouped in a cluster is called Swarm.

Multiple engines are made to run under as swarm are said to be in swarm mode. They are called worker nodes. There will be few manager nodes, who just look after these worker nodes, does allocating tasks, self-healing etc…