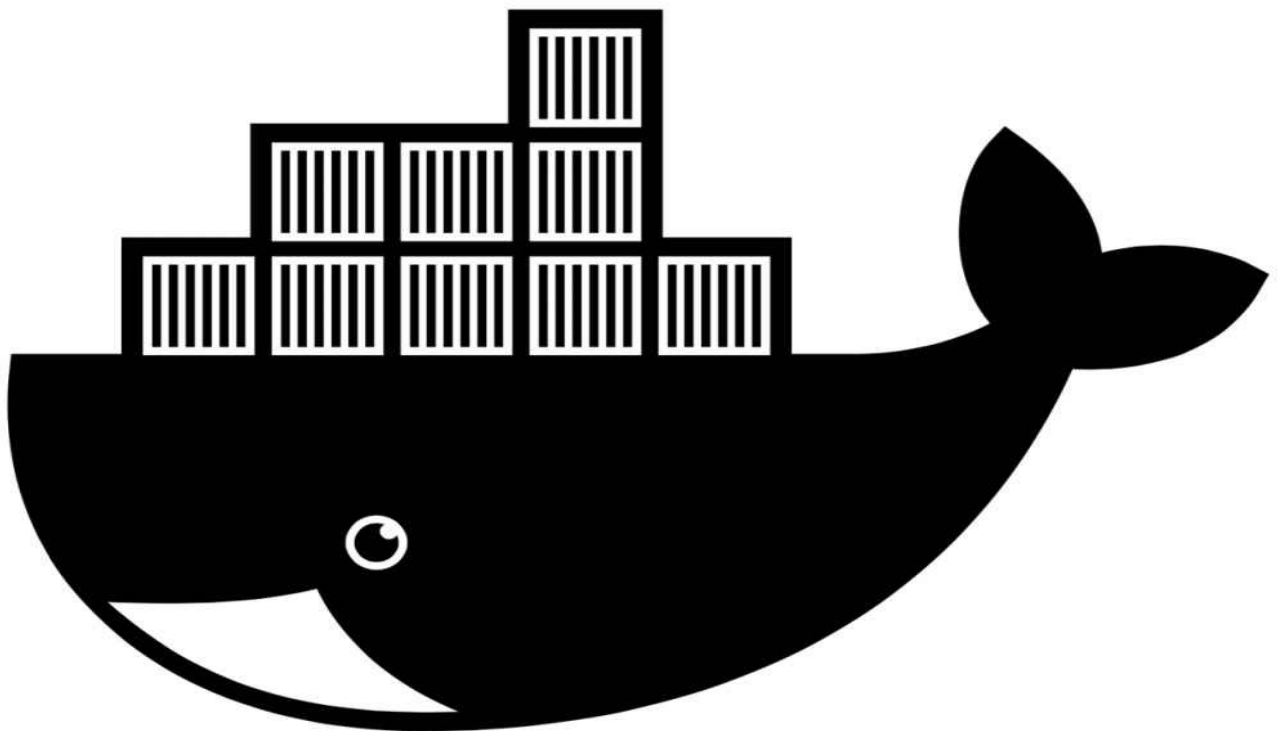


DOCKER

**The Complete Beginners Guide To Starting
With Docker**



A U S T I N S P E N C E R

Docker:

The Complete Beginners Guide to Start with
Docker

Table Of Contents

[Introduction](#)

[Chapter 1 – Getting Started with Docker](#)

[Chapter 2 – Tips to Install Docker for Mac](#)

[Chapter 3 – Understand Container, Images and Storage Drives](#)

[Conclusion](#)

Introduction

Docker helps users to package one application with all its dependencies into one standardized unit for the software development. Unlike other virtual machines, the containers don't have any high overhead and you can increase the efficiency of underlying resources and system.

It is a common practice in the industry to use VMs (virtual Machines) to run different software applications. The VMS can run these applications in the visitant operating system that runs on the virtual hardware. This hardware is powered by the OS host of the server.

VMs play an important role to provide complete procedure isolation for different applications. The problem of the host OS (operating system) can influence the software running in the operation system of guest and vice versa in different ways.

This isolation may be available at a higher cost and computational overhead disbursed virtualizing hardware for the operating system of guest is essential. Container may use dissimilar approach by leveraging the mechanics of the operating system of host and the containers offer maximum isolation of the virtual machines at one fraction of your computing power.

Rise of Docker is nothing that short of spectacular. The containers are not new technology because these also get conventional attention. The standard APIs make it easy to use containers and create a unique method for the public to collaborate all the way around libraries of the containers.

Docker plays an important role to change the face of technology landscape. In an article of "The Register", they claim that the Google runs more than two billion containers in one week.

In this book, you will read about Docker, its uses and procedure to use it. This book proves helpful for beginners to get started with Docker.

Chapter 1 – Getting Started with Docker

Docker is basically an open source and anyone can subsidize to Docker and expand it to your own needs. If you need additional features and these are not available, you can add them in the Docker.

This tool is good for system administrators and developers. It is an important part of DevOps (developers and operations) tool chains. Developers can pay attention on the writing code without thinking about system that it may ultimately be run on.

It enables them to get one head start with the use of thousands of predesigned programs to run in the Docker container. The operations staff can reduce the requirements of systems and increase their flexibility and productivity. It is a good way to reduce your overhead and footprint.

Getting Started with Docker

There are some helpful resources to get started with Docker to maintain your workflow. You will get one web-based tutorial along with one command-line simulator. It will help you to try basic commands of Docker and understand it's working.

Security and Docker

Docker increase the security of all applications running in the shared environment, but the containers may not be an alternate of security measures. You have to understand the security features of Docker to keep your containers secure. You will find about the security of Docker in last chapter. Basically Docker has three important elements, such as:

- Docker Images
- Docker Containers
- Dockerfiles

As one project, Docker offers a complete set of high-quality tools to transfer everything to make one application across machines and systems, physical or virtual, and brings loads of benefits with it.

Docker achieves full-bodied application and process and reserve containment via Containers of Linux, such as Kernel features and namespaces. The further capabilities may come from different components and parts of one project that extract the intricacy of working along with the low-level tools and APIs of Linux. These are used for the application and system management with regards to the secure containing procedures.

Main Parts of Docker Project

Project of Docker consists of different main elements and parts that all are designed on the top of existing functionality, frameworks and libraries offered by Linux Kernel or any

third party, such as aufs, mapper or LXC.

Basic Parts of Docker

- Daemon of Docker is used to manage LXC (docker) containers on its current host.
- CLI of docker is used to communicate and give command to docker daemon.
- Image index of docker is a private or public repository for the images of docker.

Elements of Docker

- Containers of docker are directories with everything about your application.
- Images of docker are snapshots of base OS (for instance, Ubuntu) or containers.
- Dockerfiles are scripts to automate the building procedure of images.

Elements of Docker

These elements are utilized by these applications making the project of the docker:

Docker Containers

This whole method of porting submissions with the use of docker depends on the container's shipment. These containers are directories that may pack (tar-archive) like others.

It can share and run crossways various platforms (hosts) and machines. You have to dependent on the host to run these containers. You have to install docker for this procedure. Containers are obtained through LXC (Linux Containers).

Linux Containers (LXC)

You can define these containers as an amalgamation of different Kernel-level features. It enables you to manage your applications and have their own environment. With the use of certain features, such as chroots, namespaces, SELinux and cgroups profiles, the LXC may contain application procedure and help with the management through restricted resources.

It may not allow you to reach beyond your file-system and restrict access to the namespace of parent. With containers, the docker makes it easy to use LXC and brings much more benefits along.

Docker Containers

The containers of docker have various main features and these features enable you to get the advantage of:

- Isolating procedure

- Application portability
- Prevent any assuaging with the external source
- Management of resource consumption

As compared to traditional virtual-machines, they require fewer resources for the deployment of isolated applications.

You are not allowed for:

- Messing with remaining procedures
- Dependency hell may be caused
- May not work on the different system
- You may be vulnerable to abuse and attacks to all resources of the system

Being depending and based on the LXC, makes one technical aspect and these containers are similar to directory, but one formatted and shaped one. This may increase the portability and gradually construct containers.

Every container may layer like one onion and every action may be taken within one container comprises of putting a separate block that actually translated to the simple change within your file system on the top of previous one. Various configurations and tools make this set-up effective in a melodious manner altogether (e.g. file-system).

This method will make containers extremely beneficial for you because you can easily create and launch new images and containers. These are kept lightweight because of layered and gradual procedure.

Everything required a file-system and take performing roll-backs and snapshots in particular times. You can get the advantage of VCS (version-control systems). Docker containers initiate from the docker image that make the base of various other layers and applications.

Docker Images

These images establish the foundation of docker container and it is a point when everything just starts to form. These are similar to the default disk images of operating system that are utilized to run different applications on desktop computers and servers.

These images will help you to get the advantage of seamless movability across systems. You can make consistent, dependable and solid base with each and everything. It is required to run all applications.

With self-contained options, the risks of system-level modifications or updates may be eliminated and the container turns out to be immune for the external exposure. This immune is important to prevent the hell dependency.

Some extra layers of applications and tools are added on the top of this base and the new images may be designed with the help of committed changes. A new container may be

created from saved things and images to continue this procedure. The file system (union-file-system) brings every layer together as one single entity and you may work with one container.

These foundation images may explicitly state the working with CLI docker to directly form one new container and they may be specified in one Dockerfile to automate image building.

Dockerfiles

These are scripts with a series of consecutive instructions, commands and directions that can be executed to make one new docker image. Every executed command is translated to one new layer of onion and makes the end product.

They can replace the procedure of undertaking everything repeatedly and manually. A Dockerfile may finish its execution and you may make one image to start one new container.

Chapter 2 – Tips to Install Docker for Mac

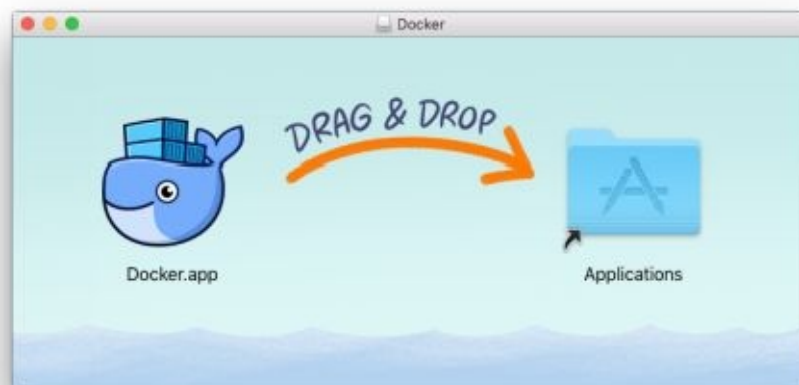
If you want to install docker, you have to check your system requirements. You can create a docker machine on Mac and copy your images and container from default machine to new docker for HyperKit VM. For Mac, you have to check these requirements:

- Your Mac should have latest model with intel hardware, MMU (memory management virtualization and EPT (extended-page tablets).
- Operating System X 10.10.3 Yosemite or latest
- Almost 4GB RAM
- VirtualBox proceeding to the version 4.3.30 should not be installed and it is incompatible with the mac for docker.

Note: If your system doesn't fulfill these requirements, you should install Toolbox of Docker that uses Virtual Box of Oracle in its place of HyperKit. The installation includes Docker CLI, Docker Engine, Docker Machine and Docker Compose.

Step 01: Run and Install Docker for Mac

Double-click the `Docker.dmg` to launch Docker installer and use whale to drag Moby to the Applications folder.



Now, you have to permit `Docker.app` using password of your system throughout the installation procedure. You will need privileged access to install the components of networking and links to the apps of Docker.

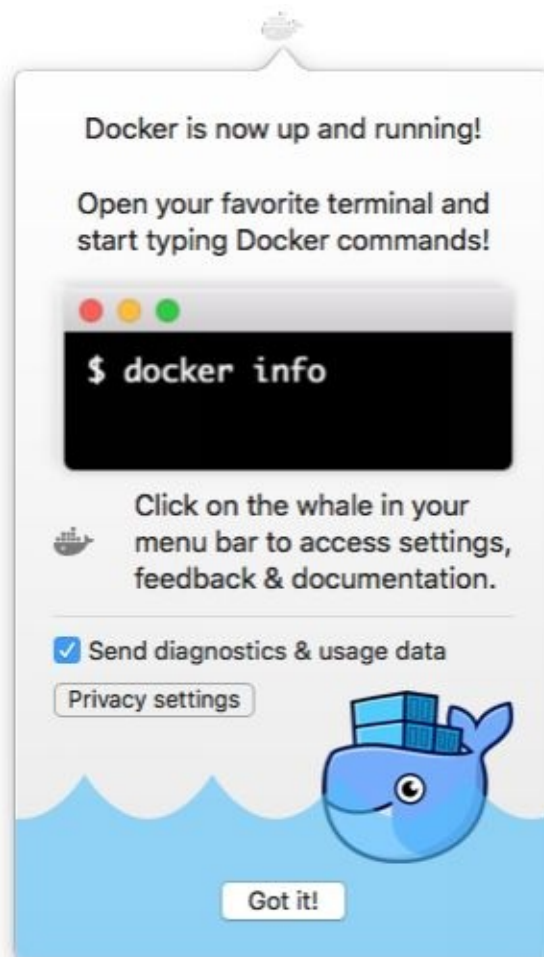
Double-click the `Docker.app` to initiate Docker.



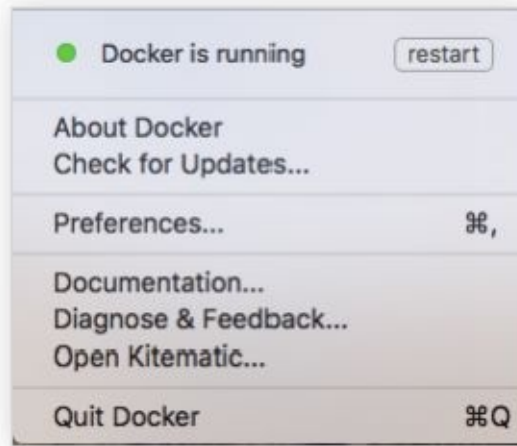
You will see a whale in the upper status bar that indicates the running status of docker, and you can easily access docker from one terminal.



After installing this app, you can get one success message and subsequent suggestions and one link for documentation. You have to click the whale (🐳) in your “status bar” to terminate this popup.



You have to tap the whale (🐳) and get some Preferences, and several other choices.



You have to choose “About Docker” to authenticate that your system has the latest version.

Well done! The latest Docker is running for Mac on your system.

Step 02: Focus on the Available Versions of the Docker Engine, Machine and Compose

You have to run these important commands in your docker to check the version, such as `docker`, `docker-compose` & `docker-machine`. Make sure these all are compatible and updated to the Docker.app.

```
$ docker --version
Docker version 1.12.0, build 8eab29e

$ docker-compose --version
docker-compose version 1.8.0, build f3628c7

$ docker-machine --version
docker-machine version 0.8.0, build b85aac1
```

Note: This example will help you to understand docker installation. Your own output can be different with a different version.

Step 03: Run Examples and Explore Applications

You can open the terminal of command-line and run a few docker commands to authenticate that the docker is working as per your expectations.

There are some commands to try, such as `docker version` is good to check the latest version, and the `docker ps` along with `docker-run-hello-world` is good to authenticate that docker is successfully running.

If you want some adventure, you can start with web server dockerized.

```
docker run -d -p 80:80 --name webserver nginx
```

If you are unable to find a local image, the docker can pull it for your assistance from a docker hub.

In your web browser, you can use `http://localhost/` to get your home page. (Meanwhile you itemized the default port of HTTP and it is not essential to affix `:80` at the final part of your URL.)



Note: Initial beta issues utilized `docker` as hostname to create the URL. The ports may be exposed on the IP addresses (private) of VM and accelerated to the local host without host name. You can see release for Beta 09.

You can run `docker ps` although the web serve may run to check the details of your web server container.

CONTAINER ID	IMAGE	COMMAND NAMES	CREATED	STATUS
56f433965490	nginx	<code>"nginx -g 'daemon off'"</code>	About a minute ago	Up About a m
inute	<code>0.0.0.0:80->80/tcp, 443/tcp</code>	webserv		

Remove or Stop Images and Containers

The webservice `nginx` will endure to run the container on the port until you remove or stop the container. Anyone who wants to stop webservice can type `docker-stop-webservice` and initiate it once again with the help of `docker-start-webservice`.

If you want to remove and stop your running container with one command, you can write `docker-rm -f-webservice`. This may remove the container, but it is not good to remove `nginx` image. If you want to list some local images, you can use `docker images`.

If you want to keep a few images, there is no need to drag them once again from the hub of docker. You can remove one image as long as you don't need it, you can use `docker-rmi <imageID>|<imageName>` for this procedure. For instance, `docker-rmi-nginx` can be the right command for you.

Preferences


You can select  —> preferences available in your menu bar. It will help you to set the

runtime options.

General



Docker is available to set for your Mac to start it automatically after your log in. You have to uncheck the option of autostart login option to avoid opening of docker session with the start of computer.

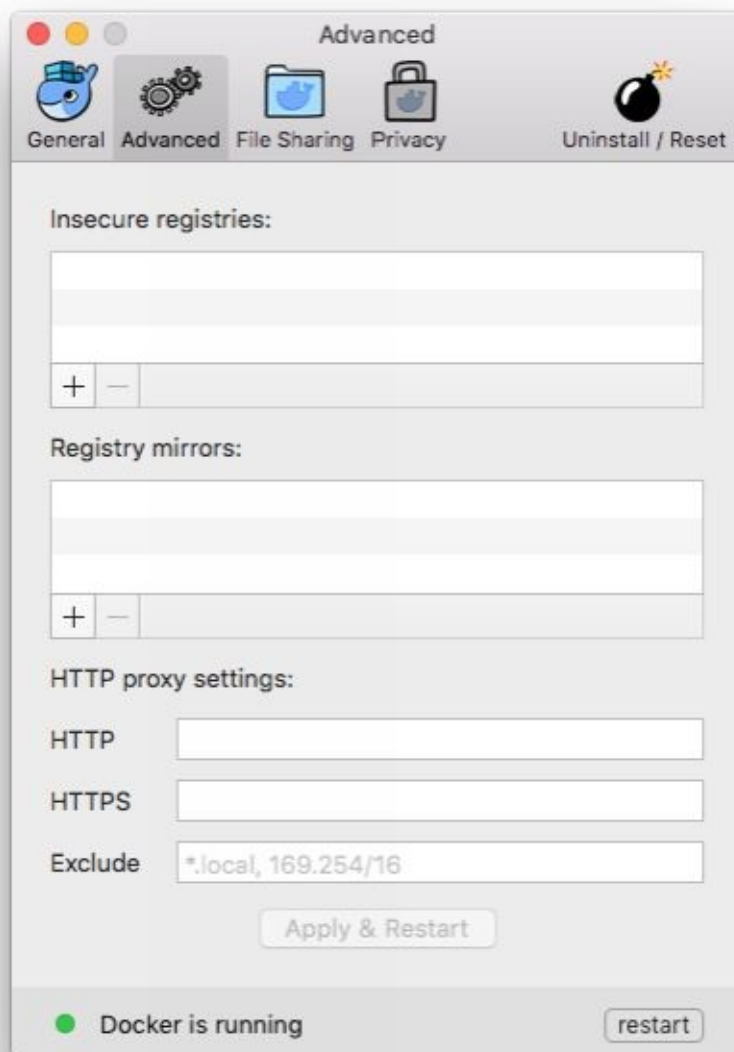
Docker is all set to check the updates and you will get notification about available updates. If the updates are found, you can click “OK” to install and accept updates. You can also cancel them to use your current version. You can also disable updates by selecting and  -> check for updates.

You can exclude the VM from the backups of your Time Machine and prevent your time Machine from docker’s backing up for the virtual machine.

The CUPs, by default, your docker for the Mac is all set to utilize two processors. It is good to increase the processing power of your app by setting it to lower or higher number to have it with docker for the use of Mac with limited computing resources.

Memory: The Docker (by default) for Mac can use 2GB runtime-memory. It is allotted from the entire accessible memory on the Mac. It is possible to increase the RAM on your app to get the advantage of quicker performance and adjust this number to higher number, such as 3 or lower, such as 1. You can adjust the memory usage of your docker.

Advanced



Addition of registries: It can be used as a substitute to use the hub of docker to store the private and public images or the trusted registry of docker. It is easy to use docker to adjust your own apprehensive registry. Users can add URLs for insecure registry mirrors and registries to host all your images.

HTTP Settings for Proxy: Docker with your mach can detect HTTPS and HTTP automatically propagate and proxy settings to docker and your containers. For instance, if you want to set proxy setting for you, you can visit <http://proxy.example.com> and the docker may use this proxy to pull containers.

Share Files

If you want to share files, you have to select directories on the Mac to share them with containers. You can click “+” to add one directory and navigate the directory that you are interested to add.



You have to hit apply and restart the available directories on the container with the use of bind mount of docker “-v” feature.

You may face some limitations on your directories and these may be shared:

- You can't make a subdirectory of a shared directory.
- They may not previously exist in docker.

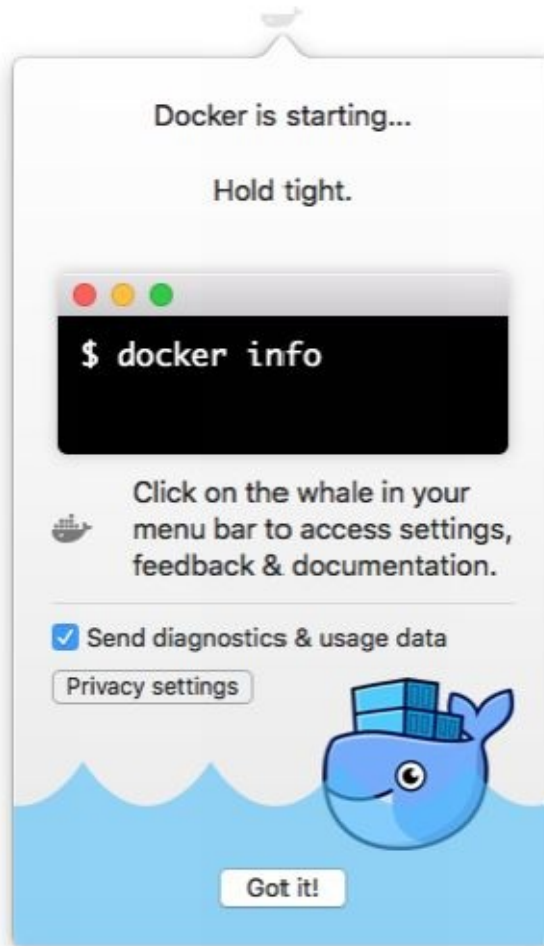
Privacy

It is easy to set docker for your mac to auto-send crash report, usage data and diagnostics. These details prove helpful for docker to bring improvements in the application and get maximum context to troubleshoot problems.


You can uncheck any option to prevent auto-sending of any data. The docker can prompt for extra information in various cases and you can enable auto-send as well.

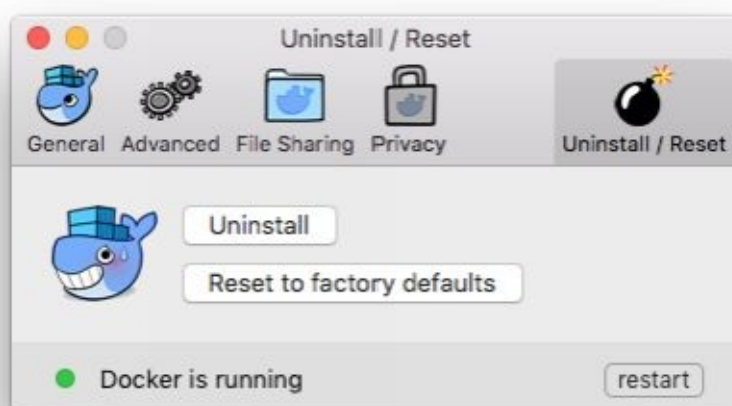


It is easy to disable and enable the auto-reporting adjustments with just one click on the popup for information, as you start your docker.



Reset or Uninstall

You can select  → options from your menu bar and hit Uninstall or Reset on the dialog of Preferences.



- Uninstall: You can select this choice to remove this docker from your system and Mac.
- Factory defaults (Reset): This option will help you to reset all your options on the docker. It will take to the initial level as you first time install docker.

You can uninstall docker for your mac by using one command-line terminal:

```
$(mdfind Docker.app)/Contents/MacOS/Docker --uninstall
```

Bash completion Installation

If you want to use bash completion, including “homebrew-bash-completion on your Mac” and bash completion script for the docker compose, docker machine and docker is easy to find in the docker.app and you can get it in resources or contents folder.

You can activate the completion of bash and these files should be symlinked or copied to the bash_completion.d directory. For instance, you can use Homebrew:

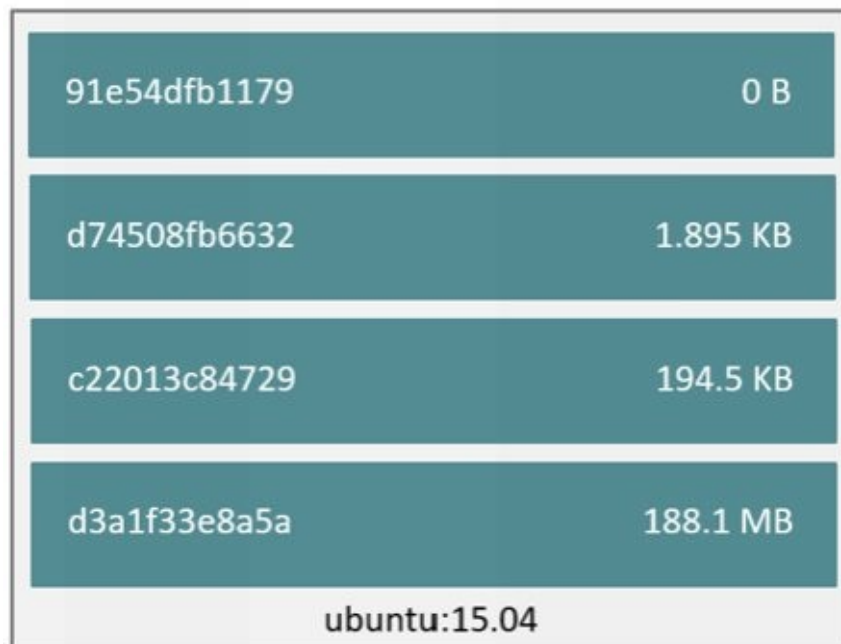
```
cd /usr/local/etc/bash_completion.d
ln -s /Applications/Docker.app/Contents/Resources/etc/docker.bash-completion
ln -s /Applications/Docker.app/Contents/Resources/etc/docker-machine.bash-completion
ln -s /Applications/Docker.app/Contents/Resources/etc/docker-compose.bash-completion
```


Chapter 3 – Understand Container, Images and Storage Drives

To effectively use your storage device, you should learn how docker creates and stores different images. You have to understand that these images will be utilized by containers. You should have short introduction to this technology to enable container and image operations.

Layers and Images

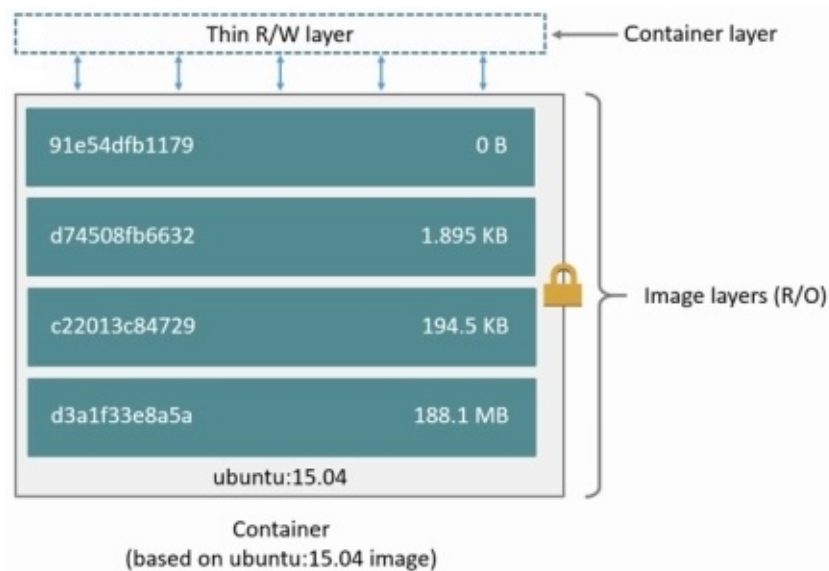
Every docker image mentions one list of layers (read-only) that represent the differences in filesystem. The layers may stack on the top of every other to make one base of one root filesystem of the container. The below diagram will help you to understand the image layers of Ubuntu 15.04:



Image

Storage driver of docker is liable to stack these layers and provide one single incorporated view.

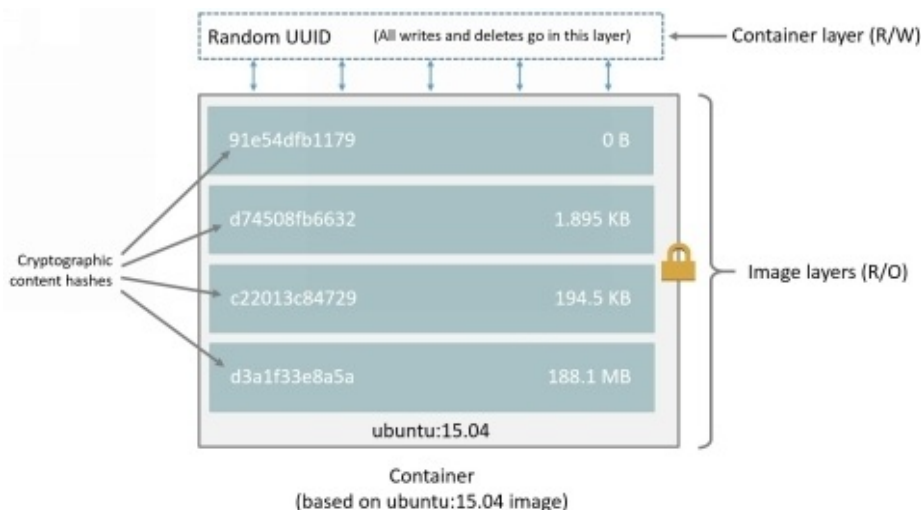
As you create one new container, you can add one new, writable and thin layer on the top of original stack. This layer may be known as container layer and all changes in your running container, such as new file writing, modification of existing files, removing files will be written on the thin layer of container. You can see the diagram to understand this procedure.



Addressable Storage for Content

The 1.10 docker is introduced with one addressable model for storage. This is a new method to work layer and image data on your disk. The layer and image data was stored and referenced with the use of UUID (randomly generated). In the unique model, you can replace it by one content hash.

This new model is good to enhance the security and offer one built-in method to avoid the collisions of ID and guarantee the integrity of data after push, load, pull and save activities. It helps you to share layers by sharing images even from a different built. See the diagram below to understand this:



You can see that all layers in the IDs are cryptographic botches and the ID container is a random UUID. There are various things to note in your new model, such as:

- Migrate current images
- Layer and image filesystem structures

Current images, pulled and created by the initial docker's versions, should be migrated before their use with the novel model. Migration may involve calculation of checksums and it is automatically performed for the first time while you start update of your docker daemon. Once the migration is finished, all tags and images will become new and secure IDs.

This migration procedure is transparent and automatic and it is intensive on a computational level. It means you have to take some time with image data. During migration time, the docker daemon may not give proper response to requests.

You can get the advantage of migration tool to migrate current images to new format before upgrading your daemon docker. The upgraded daemons docker will not have to perform the in-band migration and avoid any linked downtime. It offers one way to physically migrate current images to distribute them to other daemons docker in current environment with recent docker's version.

This tool is offered by Docker and it work as one container. You can use <https://github.com/docker/v1.10-migrator/releases> to download it.

If you are running migrator image, you have to expose the data of directory's host to your container. If you have to use default data path of docker, the command for docker can be this one:

```
$ sudo docker run --rm -v /var/lib/docker:/var/lib/docker docker/v1.10-migrator
```

If you want to use devicemapper driver for storage, you have to write `--privileged` choices to give access to storage driver to your container.

Example of Migration

Check the example below to use the migration tool on docker 1.9.1 version and AUFS drivers for storage. This host may run on one t2.micro AWS EC2 along with 8GB SSD volume, 1GB RAM and 1 vCPU. The data directory (`/var/lib/docker`) of docker consumed almost 2GB space.

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
jenkins              latest             285c9f0f9d3d       17 hours ago      708.5 MB
mysql                latest             d39c3fa09ced       8 days ago        368.3 MB
mongo                latest             a74137af4532       13 days ago       317.4 MB
postgres             latest             9aae83d4127f       13 days ago       270.7 MB
redis                latest             8bccd73928d9       2 weeks ago       151.3 MB
centos                latest             c8a648134623       4 weeks ago       196.6 MB
ubuntu                15.04             c8be1ac8145a       7 weeks ago       131.3 MB

$ sudo du -hs /var/lib/docker
2.0G   /var/lib/docker

$ time docker run --rm -v /var/lib/docker:/var/lib/docker docker/v1.10-migrator

Unable to find image 'docker/v1.10-migrator:latest' locally
latest: Pulling from docker/v1.10-migrator
ed1f33c5883d: Pull complete
b3ca410aa2c1: Pull complete
2b9c6ed9099e: Pull complete
dce7e318b173: Pull complete
Digest: sha256:bd2b245d5d22dd94ec4a8417a9b81bb5e90b171031c6e216484db3fe30ec2097
Status: Downloaded newer image for docker/v1.10-migrator:latest
time="2016-01-27T12:31:06Z" level=debug msg="Assembling tar data for 01e70da302a553ba13485ad020a0d77db47575a31c4f48221137bb08f45878d from /var/lib/docker/aufs/diff/01e70da302a553ba13485ad020a0d77db47575a31c4f48221137bb08f45878d"
time="2016-01-27T12:31:06Z" level=debug msg="Assembling tar data for 07ac220aeef9feb1ac16a9d1a4ef7ef3c8cbf5ed0be6b6f4c35952ed7920d from /var/lib/docker/aufs/diff/07ac220aeef9feb1ac16a9d1a4ef7ef3c8cbf5ed0be6b6f4c35952ed7920d"
<snip>
time="2016-01-27T12:32:00Z" level=debug msg="layer dbacfa057b30b1feaf15937c28bd8ca0d6c634fc311ccc35bd8d56d017595d5b took 10.80 seconds"

real    @m59.583s
user    @m0.046s
sys     @m0.008s
```

With the help of time Unix command, the docker can produce time for a particular operation. If you want to migrate seven images taking 2GB space on your disk, it will take almost one minute. This may include the time required to pull the `docker/v1.10-migrator` image in almost 3.5 seconds. This similar operation on the m4.10 by large EC2 with 160GB RAM, 8GB EBS and provisioned IOPS and 40 vCPUs may take different time. See below to improve this operation:

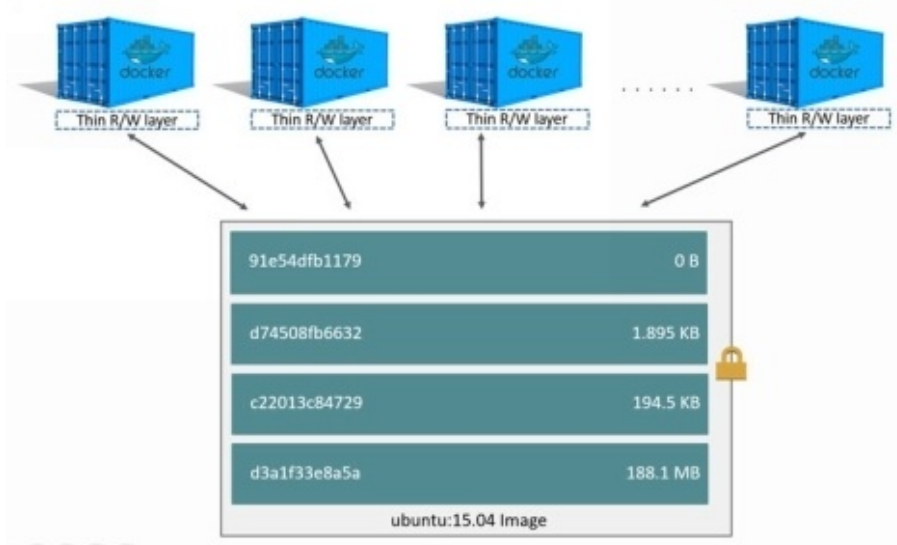
```
real    0m9.871s
user    0m0.094s
sys     0m0.021s
```

It is enough to see the effect of hardware spec on the migration operation.

Layers and Containers

The basic different between one image and one container is writable layer on the top. All writes to your container may add one new or modify prevailing data stored in the writable layer. If you delete your container, it will also delete your writable layer. There will be no changes in the original image.

Every container contains its individual writable and thin layer and all changes will be secured in the layer of container. It means multiple containers can share the accessibility to the similar image and have their own state of data. See the diagram of multiple containers:



The storage drive in the docker is responsible to manage and enable both writable layers of container and image layers. The storage driver can accomplish between drives in a different way. There are 2 main technologies behind the container management and image of docker. The docker layer stackable images with CoW (copy-on-write) ability.

Strategy of Copy-on-Write

You can optimize resources by sharing them and people often do this impulsively in their life. For instance, Joseph and Jane are twins taking calculus classes at separate times from

separate teachers. They can share their exercise book by passing it to each other. Jane has to complete the homework on the 11th page his book. Now, the original exercise book can't be changed and only Jane can copy this page.

Copy-on-write strategy is simple for copying and sharing. This strategy enables system (that requires similar data) processes the data instead of getting their own copy. At a particular point, if one procedure requires some modification to write data, only the procedure that should be written will copy the data. All other procedures will be continuing to the use of actual data.

Docker requires one copy-on-write technology with containers and image. This strategy can optimize the performance of your container and disk space. In the next section, the system will work to leverage the copy with containers and images via copying and sharing.

Promote Small Images via Sharing

You have to look at the CoW and image layers technology. All images have layers in the local storage of docker and managed by the driver of storage. The Linux-based docker may host this under `/var/lib/docker/`. These clients often report the images layers and the below command will prove helpful for you:

```
$ docker pull ubuntu:15.04

15.04: Pulling from library/ubuntu
1ba8ac955b97: Pull complete
f157c4e5ede7: Pull complete
0b7e98f84c4c: Pull complete
a3ed95caeb02: Pull complete
Digest: sha256:5e279a9df07990286cce22e1b0f5b0490629ca6d187698746ae5e28e604a640e
Status: Downloaded newer image for ubuntu:15.04
```

You can notice output because these commands can actually grab four image layers. Every line is listing the layer images and UUID has. Combination of these layers will help you to make your favorite images.

Every layer will be stored in the directory in the host of docker. It may use local storage area. Earlier versions of docker were storing every layer in one direction with similar name as the actual name of image layer. If you are using 1.9.1 version of docker, you can apply this command and get desired results. See the instruction below:

```

$ docker pull ubuntu:15.04

15.04: Pulling from library/ubuntu
47984b517ca9: Pull complete
df6e891a3ea9: Pull complete
e65155041eed: Pull complete
c8belac8145a: Pull complete
Digest: sha256:5e279a9df07990286cce22e1b0f5b0490629ca6d187698746ae5e28e604a640e
Status: Downloaded newer image for ubuntu:15.04

$ ls /var/lib/docker/aufs/layers

47984b517ca9ca0312aced5c9698753ffa964c2015f2a5f18e5efa9848cf30e2
c8belac8145a6e59a55667f573883749ad66eaeef92b4df17e5ea1260e2d7356
df6e891a3ea9c9dce2a388a2cf1b1711629557454fd120abd5be6d32329a0e0ac
e65155041eed7ec58dea78d90286048055ca75d41ea893c7246e794389ecf203

```

Check the matching procedure of four directories with IDs layer of downloaded images. You can compare with similar operations performed on the host version 1.10 of docker.

```

$ docker pull ubuntu:15.04
15.04: Pulling from library/ubuntu
1ba8ac955b97: Pull complete
f157c4e5ede7: Pull complete
0b7e98f84c4c: Pull complete
a3ed95caeb02: Pull complete
Digest: sha256:5e279a9df07990286cce22e1b0f5b0490629ca6d187698746ae5e28e604a640e
Status: Downloaded newer image for ubuntu:15.04

$ ls /var/lib/docker/aufs/layers/
1d6674ff835b10f76e354806e16b950f91a191d3b471236609ab13a930275e24
5dbb0cbe0148cf447b9464a358c1587be586058d9a4c9ce079320265e2bb94e7
bef7199f2ed8e86fa4ada1309cfad3089e0542fec8894690529e4c04a7ca2d73
ebf814eccfe98f2704660ca1d844e4348db3b5cc637eb905d4818fbfb00a06a

```

You can see matchup procedure of all four directories with the layer IDs of images.

You can notice different among image management in afore and after versions. All docker's version enables images to carefully share the layers. For instance, you can grab one image to share some similar layers of image as one image and it may be already pulled. The daemon Docker can recognize this and pulls the required layers out of their stored location. The second pull proves helpful to pull images with common features and layers.

This illustration can help you, just start with 15.04 Ubuntu image that you have recently pulled and make some changes into it to build a new image. You can use docker build or dockerfile command to make your work easy.

Take empty directory and create one simple dockerfile to start with 15.04 ubuntu image.

```
FROM ubuntu:15.04
```

You can add one new file and it will be known as new files in the /tmp directory of images and with a line "Hello World". Once you have done with it, the dockerfile will have these two lines:


```
FROM ubuntu:15.04

RUN echo "Hello world" > /tmp/newfile
```

You have to close and save file and from the terminal in similar folder as Dockerfile, you can run the given commands:

```
$ docker build -t changed-ubuntu .

Sending build context to Docker daemon 2.048 kB
Step 1 : FROM ubuntu:15.04
---> 3f7bcee56709
Step 2 : RUN echo "Hello world" > /tmp/newfile
---> Running in d14acd6fad4e
---> 94e6b7d2c720
Removing intermediate container d14acd6fad4e
Successfully built 94e6b7d2c720
```

Note: The (.) period is available at the end of above command and this period is important. It will communicate with docker-build command to utilize the current directory and build the context.

The above shown output will show you the new image and its ID will be 94e6b7d2c720.

Execute the docker-images command to authenticate the changed-ubuntu image in the local storage area of docker host.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
changed-ubuntu	latest	03b964f68d06	33 seconds ago	131.4 MB
ubuntu	15.04	013f3d01d247	6 weeks ago	131.3 MB

You have to run the history of docker command to check the layers of images used to create changed-ubuntu pictures.

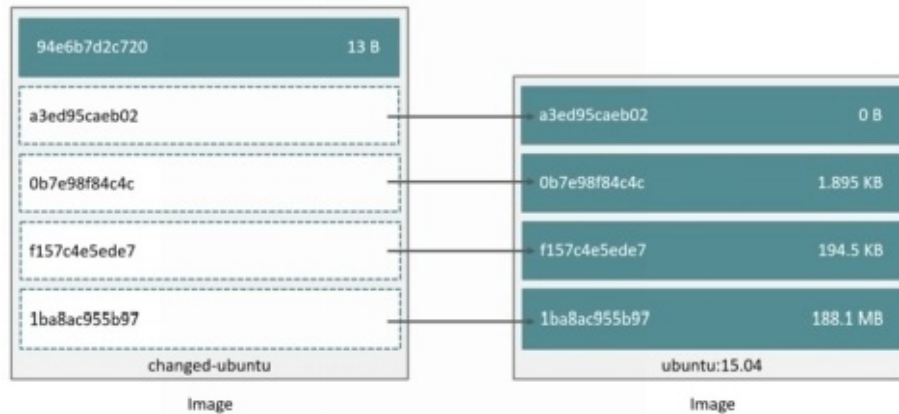
```
$ docker history changed-ubuntu
IMAGE          CREATED          CREATED BY                                      SIZE
E             COMMENT
94e6b7d2c720   2 minutes ago   /bin/sh -c echo "Hello world" > /tmp/newfile  12 B
3f7bcee56709   6 weeks ago     /bin/sh -c #(nop) CMD ["/bin/bash"]         0 B
<missing>      6 weeks ago     /bin/sh -c sed -i 's/"#\s*"$(deb.*universe)"/  1.87
9 kB
<missing>      6 weeks ago     /bin/sh -c echo '#!/bin/sh' > /usr/sbin/polic  701
B
<missing>      6 weeks ago     /bin/sh -c #(nop) ADD file:8e4943cd86e9b2ca13  131.
3 MB
```

The history of docker output reveals the new image 94e6b7d2c720 layer at top. This new layer of image is easy to add because it is created by the tmp/newfile of hello world in the dockerfile. The four image layers under it are accurately similar image layers to make Ubuntu: 15.04 pictures.

Note: under the addressable storage content model familiarized with 1.10 docker and

history data of image will not be stored in the configuration file with every image layer. It is stored as one string of text in the single configuration to whole image. This may result in the similar image layer and it will be shown mixing in the docker-history output command. This behavior is normal and you can ignore it. These types of images are commonly known as level images.

You can check the changed-ubuntu image because it doesn't have an individual copy of each layer. You can see in the below diagram that the image is shared with four layers with 15.04 ubuntu image:

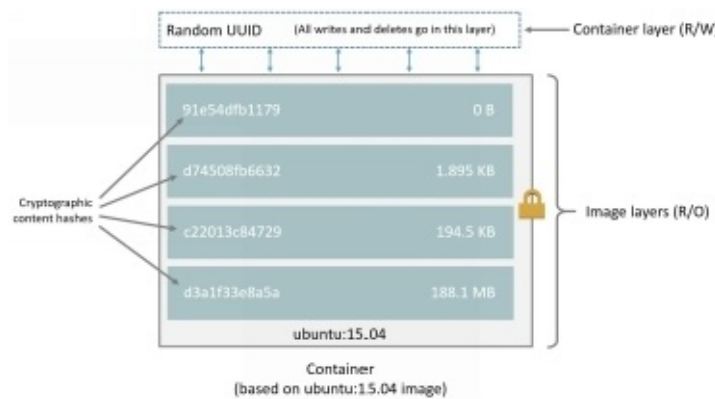


The docker-history command will display the size of every layer of image. You can notice the 94e6b7d2c720 layer consumes only 12 bytes space on the disk. It means that the image of changed-ubuntu requires only 12 bytes extra space on the docker. All layers already exist on the host docker and shared by various other images.

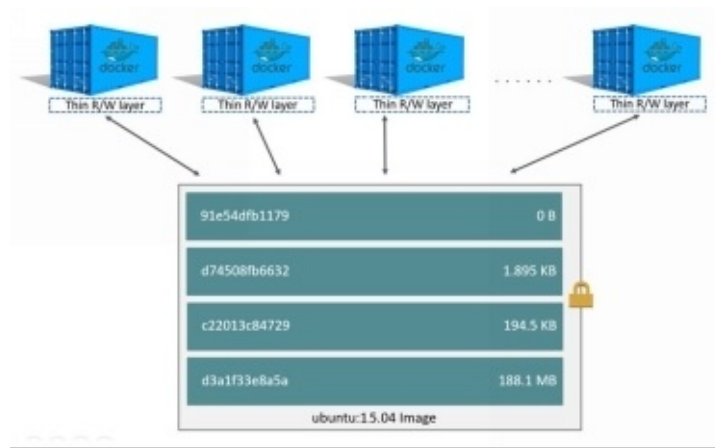
Sharing image layers make the images of containers and docker really space efficient.

Copying can Make the Containers Efficient

Container is one docker image with one thin writable and layer of container. This diagram will help you to understand the layers of container on the basis of 15.04 ubuntu image:



All writes on the container are secured in one writable layer of container. The other layers are only RO (read only) layers and you can't change them. You can use multiple containers to share one single foundation image. The diagram below will help you to understand the sharing of images by multiple containers. Every container may possess its individual layers. See the image:

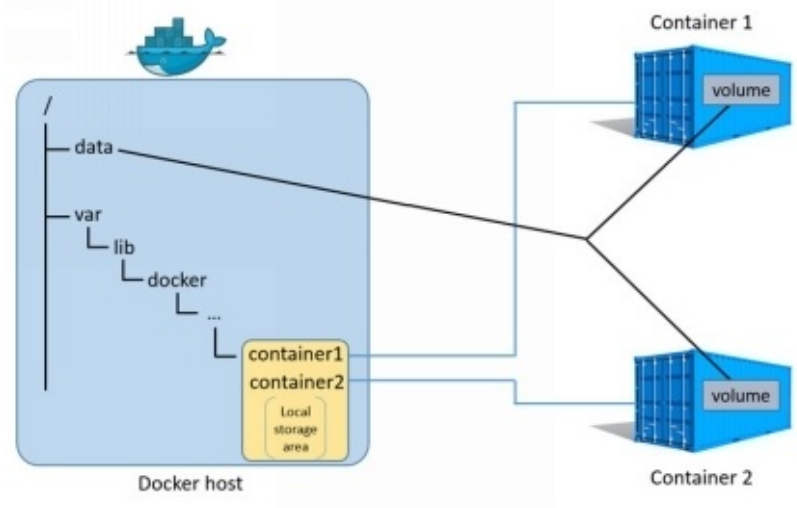


As you modify the existing file in a container, the docker utilizes your storage driver to perform operation of CoW. This may specify the operations on the basis of storage driver. For OverlayFS and AUFS storage drivers, the CoW operations can be as follows:

- You can search through the layers of image to update file. This procedure starts at the top of your new layer and work in the downward direction to the foundation layer. It will work on one layer at a time.
- You can perform one “copy-up” operation on initial copy of your file. The copy up may copy your file up to the individual container with thin and writable layer.
- You can modify the copy of file in thin writable layer of your container.

ZFS, Btrfs and many other drivers can handle the CoW (copy-on-write) contrarily.

Storage Drive and Data Volumes



As you delete one container, the data written to this container may not be stored in the data volume is also deleted with this container. The data volume is one file or directory in the filesystem mounted directly in one container. These volumes will not be controlled by any storage driver.

Data volumes live exterior of local storage on the host docker. It can reinforce the independence from the control of storage driver. As one container is deleted, the data stored in the volumes persists on the host docker.

Conclusion

Docker is designed for different applications and container. These containers are good for developers to develop codes and make their own assignments. There are different tools in docker for your assistance and you can use these tools to create different things.

This open source program is good to automate the deployment of software and container with extra layer of abstraction and automation. In this book, you will find useful details about docker and containers.