H3ABioNet

Pan African Bioinformatics Network for H3Africa

Docker: GETTING STARTED

A hands-on step-by-step basic guide to Docker essentials.

Developed by

The H3ABionet Pipelines and Computing Work Package,
Computing Infrastructure project team

Prepared for the greater

H3ABioNet and H3Africa Consortium communities

Document Control

Date	Version	Notes
	1.0	Initial guide development

Project Members

Last Name	First Name	Institution	Country
Ghanmi	Nidhal	Institute of Pasteur , Tunis	Tunisia
Lukyamuzi	Edward	Uganda Virus Research Institute	Uganda
Maslamoney	Suresh	Computational Biology Division, University of Cape Town	South Africa
Meintjes	Ayton	Computational Biology Division, University of Cape Town	South Africa
Oloyede	Emmnauel	National Biotechnology Development Agency	Nigeria
Wamala	Timothy	Uganda Virus Research Institute	Uganda

Background

This guide is produced by the Computing & Infrastructure working group under the Pipelines and Computing work package. It is intended to serve as a self-sufficient support guide on all major aspects around docker technology for system administrators across the H3ABioNet and its collaborating partners. All commands and screenshots are based on the Ubuntu 20.04 Operating System (OS). The guide has been designed to be effective as a standalone self-guided walkthrough of installing, deploying and management of docker containers- while the guide is written in an easy to read, user-friendly way, it assumes familiarity with navigating around an Ubuntu 20.04 Linux system. If you are not familiar with Linux, refer to our "LINUX: GETTING STARTED" guide on the https://h3abionet.org/tools-and-services/technical-guidelines website. Following the instructions in this guide will give the reader hands on experience with installing the Docker community edition, create a basic Docker container with a static web page. It includes useful commands to manage multiple containers and images.

It is recommended that the reader first reads this guide in its entirety before following the step by step instructions.

This guide makes the following assumptions:

- The reader is comfortable navigating the Ubuntu 20.04 Linux OS
- Is able to install software and edit files at the command line

How to read this guide

- General text describing each section and commands are written in the Arial font, size 11. This text is to be read to understand which skills will be gained in the following section and to describe the command operation.
- When noting a variation to a default command or to note a point or warning, the default text will be highlighted in yellow.
- When noting a tip, for example, a command that can be run in multiple ways producing the same output, this text will be highlighted in green.
- When giving examples of actual expected output, this text will be highlighted with a grey background.
- When listing a command that is to be run by the reader to produce an output, the command will be in light blue italics.

Abbreviations

Abbreviation	Description
OS	**Operating System**
cli	**Command line interface**
gui	**Graphical user interface**
dhcp	**Dynamic host configuration protocol**
IP	**Internet Protocol**
VM	Virtual Machine

1. What are Containers?

In the recent past, the industry standard was to use Virtual Machines (VMs) to run software applications. VMs run applications inside a guest Operating System, which runs on virtual hardware powered by the host server's OS.

VMs are great at providing full process isolation for applications: there are very few ways a problem in the host operating system can affect the software running in the guest operating system, and vice-versa. But this isolation comes at great cost — the computational overhead spent virtualizing hardware for a guest OS to use is substantial.

Containers take a different approach, by leveraging the low-level mechanics of the host operating system, containers provide most of the isolation of virtual machines at a fraction of the computing power.

Essentially, Docker is a container-based system for your applications. If you're used to the concept of virtual servers, Docker provides further levels of abstraction for your application. Here's a visual representation of how containers differ from VMs:

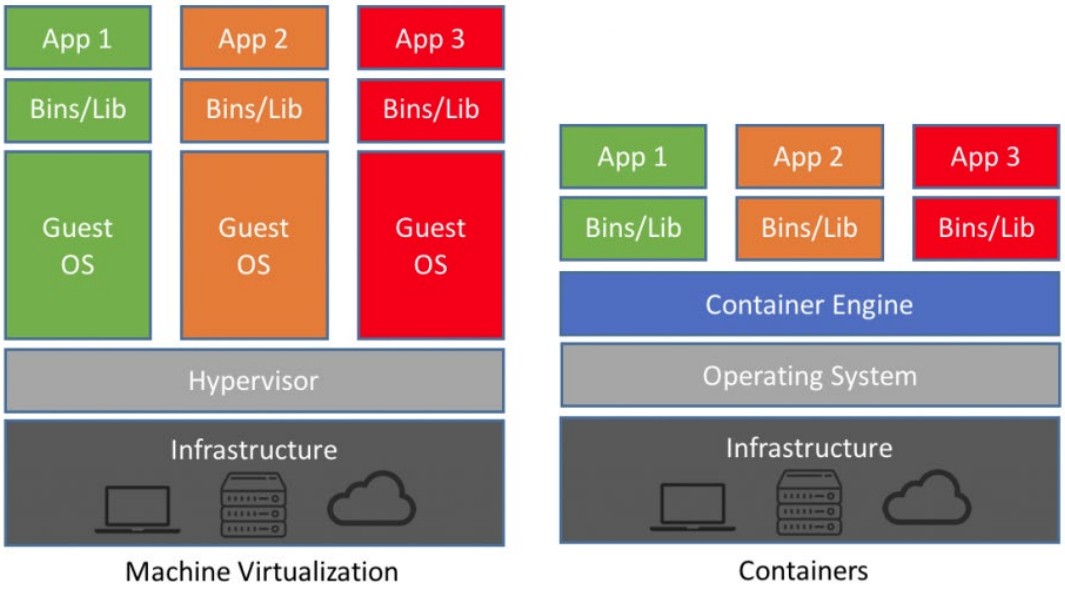

Image Source: https://blog.netapp.com/

What's the Diff: VMs vs Containers

VMs	Containers
Heavyweight	Lightweight
Limited performance	Native performance
Each VM runs in its own OS	All containers share the host OS
Hardware-level virtualization	OS virtualization
Startup time in minutes	Startup time in milliseconds
Allocates required memory	Requires less memory space
Fully isolated and hence more secure	Process-level isolation, possibly less secure

1.1 Types of Containers

Linux Containers (LXC) — The original Linux container technology is Linux Containers, commonly known as LXC. LXC is a Linux operating system level virtualization method for running multiple isolated Linux systems on a single host.

Docker — Docker started as a project to build single-application LXC containers, introducing several changes to LXC that make containers more portable and flexible to use. It later morphed into its own container runtime environment. At a high level, Docker is a Linux utility that can efficiently create, ship, and run containers.

1.1.1 Popular Container Providers

1. Linux Containers
   - LXC
   - LXD
   - CGManager
2. Docker
3. Singularity
4. Windows Server Containers

1.2 What is Docker

Unless you've been living without internet access for the last years, it would be hard not to at least have heard of Docker. But as an emerging technology, not everyone has taken the time to work out what Docker is, where it fits in, and how it can benefit you.

So, what exactly is Docker? Here's how Docker themselves describe it:

"Docker is an open platform for developers and sysadmins of distributed applications."

Wikipedia defines Docker as:

"an open-source project that automates the deployment of software applications inside containers by providing an additional layer of abstraction and automation of OS-level virtualization on Linux."

Wow! That's a mouthful. In simpler words, Docker is a tool that allows developers, sysadmins etc. to easily deploy their applications in a sandbox (called containers) to run on the host operating system i.e. Linux. The key benefit of Docker is that it allows users to package an application with all of its dependencies into a standardized unit for software development. Unlike virtual machines, containers do not have high overhead and hence enable more efficient usage of the underlying system and resources.

1.3 When to Use Docker

If your application fits into one or more of the following categories, Docker may be a good fit:

Learning new technologies: To get started with a new tool without spending time on installation and configuration, Docker offers an isolated and disposable environment. Many projects maintain Docker images with their applications already installed and configured.

Basic use cases: Pulling images from Docker Hub is also a good solution if your application is basic or standard enough to work with a default Docker image. Cases such as hosting a website using a LAMP

stack or using a reverse proxy have an official or well-supported image available on DockerHub. If the default configuration in these images is acceptable for your needs, then pulling the image can save a lot of time that would otherwise be spent setting up your environment and installing the necessary tools.

App isolation: If you want to run multiple applications on one server, keeping the components of each application in separate containers will prevent problems with dependency management.

Developer teams: If you have developers working with different setups, Docker provides a convenient way to have local development environments that closely match the production environment, without needing to ssh into a remote box.

1.4 When Not to Use Docker

There are also times when Docker isn't the best solution. Here are some examples:

Your app is complicated and you are not/do not have a sysadmin. For large or complicated applications, using a pre-made Dockerfile or pulling an existing image will not be sufficient. Building, editing, and managing communication between multiple containers on multiple servers is a time-consuming task.

Performance is critical to your application. Docker shines compared to virtual machines when it comes to performance because containers share the host kernel and do not emulate a full operating system. However, Docker does impose performance costs. Processes running within a container will not be quite as fast as those run on the native OS. If you need to get the best possible performance out of your server, you may want to avoid Docker.

Security is critical to your application. As mentioned above, keeping the different components of an application in separate containers provides some security benefits, since a compromise in one container can't easily affect the rest of your system. However, Docker's containerization approach raises its own security challenges, especially for more complicated applications. These issues are solvable but require attention from an experienced security engineer.

Multiple operating systems. Since Docker containers share the host computer's operating system, if you want to run or test the same application on different operating systems, you will need to use virtual machines instead of Docker.

Clusters. Docker containers on separate servers can be combined to form a cluster with Docker Swarm. However, Docker does not take the place of provisioning or automation tools such as Ansible, SaltStack, and Chef. In addition, Docker support Kubernetes, hinting that Docker Swarm may not be sufficient as a stand-alone cluster manager.

1.5 Core Components of Docker

Docker Engine is one of the core components of Docker. It is responsible for the overall functioning of the Docker platform.

Docker Engine is a client-server based application and consists of 3 main components.

- Server
- REST API
- Client

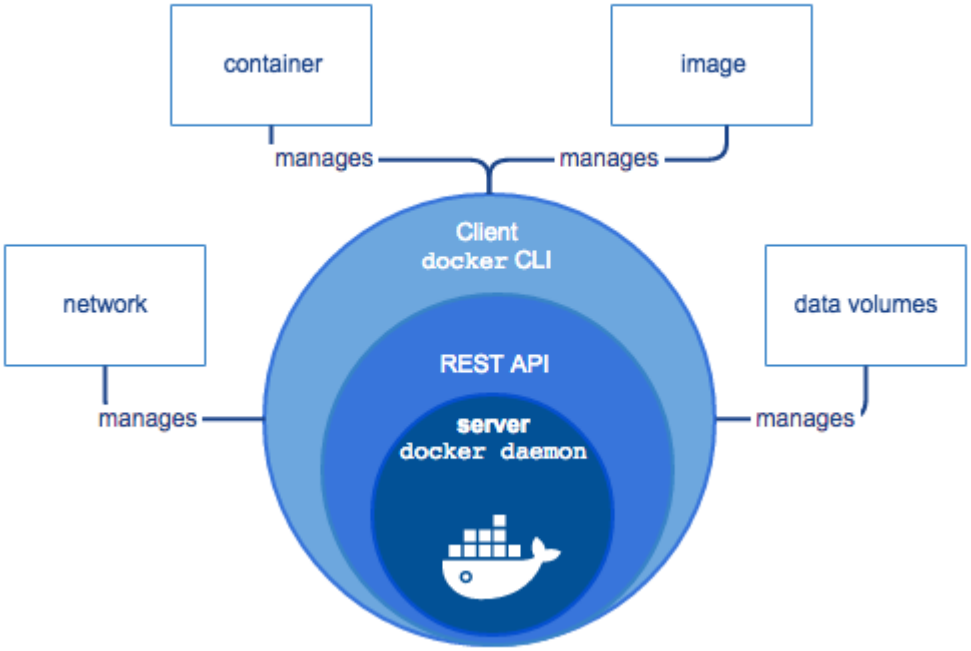

Image Source: https://docs.docker.com

The **Server** runs a daemon known as dockerd (Docker Daemon), which is nothing but a process. It is responsible for creating and managing Docker Images, Containers, Networks and Volumes on the Docker platform.

The **REST API** specifies how the applications can interact with the Server, and instruct it to get their job done.

The **Client** is nothing but a command line interface, that allows users to interact with Docker using the commands.

1.6 Docker Terminology

Let us take a quick look at some of the terminology associated with Docker.

Docker Images and Docker Containers are the two essential things that you will come across daily while working with Docker.

In simple terms, a **Docker Image** is a template that contains the application, and all the dependencies required to run that application on Docker.

On the other hand, as stated earlier, a **Docker Container** is a logical entity. In more precise terms, it is a running instance of the Docker Image.

What is Docker Hub?

Docker Hub is the official online repository where you could find all the Docker Images that are available for us to use.

Docker Hub also allows us to store and distribute our custom images as well if we wish to do so. We could also make them either public or private, based on our requirements.

1.7 Docker Editions

Docker is available in 2 different editions, as listed below:

- Community Edition (CE)
- Enterprise Edition (EE)

The Community Edition is suitable for individual developers and small teams. It offers limited functionality, in comparison to the Enterprise Edition.

The Enterprise Edition, on the other hand, is suitable for large teams and for using Docker in production environments.

The Enterprise Edition is further categorized into three different editions, as listed below:

- Basic Edition
- Standard Edition
- Advanced Edition

2. Installing Docker CE on Ubuntu 20.04/20.04 LTS

Let's explore the use of the Advanced Packaging Tool (APT) and Snap to install docker on Ubuntu 20.04 LTS. In recent years a new package management system called Snap has been under development by the Ubuntu team at Canonical, Ltd. Although there are no official plans to replace APT entirely with Snap, the list of packages that can now be installed as “snaps” continues to grow

2.1. Install Using Advanced Packaging Tool (APT)

First, update your existing list of packages: the system needs to be updated to make it safer and reliable to install Docker.

```
$ sudo apt update
```

Install Prerequisite Packages

Once we have updated the system, we need to install some necessary packages before we are ready to install Docker.

```
$ sudo apt install curl apt-transport-https ca-certificates software-properties-common
```

Add the Docker Repositories

This enables us to use the officially supported method of the installation making the installation process much easier.

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

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

Then, 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 update

Make sure you are installing from the Docker repo instead of the default Ubuntu repo with this command:

$ apt-cache policy docker-ce

Install Docker: Use the apt command

$sudo apt install docker-ce

$ sudo service docker start

Check the docker status: good idea to check that it's running

$ sudo systemctl status docker

Add Docker service to the system startup: so that it will start automatically on system boot.

$ sudo systemctl enable docker

Check the docker version

$ docker --version

Optional : Add current user to docker group to avoid using sudo

$ sudo usermod -aG docker ubuntu

Give Permissions to current user to execute docker binary

$ sudo chmod 755 /usr/bin/docker

2.2 Install Using Snap

Snap packages are universal Linux packages that make it easy to deploy applications/software on any Linux distribution.

Let's see how to manage Docker installation using Snap
First step is to check versions available for Docker in Snap packages

```
$ snap info docker
```

Next is to install docker using one of the channels (deployment packages) available

```
$ sudo snap install docker
```

Verify that docker has been installed

```
$ snap services docker
```

It prints service name as **'docker.dockerd'** that is different from service name when you deploy without using Snap package

Start docker

```
$ sudo snap start docker
```

Stop docker installed using Snap package

```
$ sudo snap stop docker
```

To uninstall or remove docker installed using Snap package

```
$ sudo snap remove docker
```

3. Getting Started

3.1 Basic Commands

Info command: gives information about the docker setup on your machine/vm

```
$ docker info
```

Run Container and enter its shell

```
$ sudo docker run -i -t ubuntu /bin/bash
```

This should give you a new command prompt inside the container, very similar to if you had ssh'ed into a remote machine. In this case the flags -i and -t tell Docker we want an interactive session with a tty attached. The command /bin/bash gives a bash shell. When you exit the shell the container will stop — containers only run as long as their main process

```
$ docker run ubuntu echo hello-world
hello-world
```

Run Container and enter its shell: use -h command line parameter to specify a container name.

```
$ docker run -h CONTAINER1 -i -t ubuntu /bin/bash
```

Output of the command above will open a tty inside the container

```
root@CONTAINER1:/#
```

Run Container with Networking mode: use the flag *--net*

```
$ docker run -h CONTAINER2 -i -t --net="bridge" ubuntu /bin/bash
```

List of docker containers running

```
$ docker ps -a
```

Inspect a Container

```
$ docker inspect CONTAINER_NAME
```

Start a Stopped Container

```
$ docker start CONTAINER_NAME
```

Enter the Shell of a Started Container

```
$ docker attach CONTAINER_NAME
```

where CONTAINER_NAME is the container name.

Delete a Container

```
docker rm CONTAINER_NAME
```

Detach from a Container

```
docker run -t -i → can be detached with ^P^Q and reattached with docker attach
docker run -i    → cannot be detached with ^P^Q; will disrupt stdin
docker run       → cannot be detached with ^P^Q;
                   can SIGKILL client; can reattach with docker attach
```

Docker Logs: get a list of commands executed in the container.

```
$ docker logs CONTAINER_NAME
```

Pause a Container: Note : Container needs to be in the Started Phase

```
$ docker pause CONTAINER_NAME
```

UnPause a Paused Container

```
$ docker unpause CONTAINER_NAME
```

Removing all the Containers

```
$ docker rm `docker ps --no-trunc -aq`
```

List Docker Networks

```
$ docker network ls
```

Rename a Docker Container

```
$ docker rename OLD_NAME NEW_NAME
```

4. Docker Images

Show images

```
$ sudo docker images
```

Specifying a Variant

```
$ sudo docker run -i -t ubuntu:20.04 /bin/bash
```

Pull an Image

```
$ sudo docker pull ubuntu
```

Create your own image

Make a docker file directory

```
$ mkdir docker-file
```

Make a docker file

```
$ cd docker-file/
$ touch Dockerfile
```

Add content to docker file

```
$ cat > Dockerfile
FROM ubuntu:20.04
MAINTAINER xyz "xyz@xyz.com"
RUN apt-get update
RUN apt-get install -y nginx
RUN echo 'Our first Docker image for Nginx' > /usr/share/nginx/html/index.html
EXPOSE 80
```

Ctrl+D to save the changes

Creating a image

Build a docker image

```
$ sudo docker build -t="test/my_nginx"
```

Check that image has been created

```
$ docker images | grep nginx
test/my_nginx
```

Removing a Docker Image

```
$ docker rmi test/my_nginx
```

5. Docker Registry

Docker allows to bundle and push the customized images on the docker hub or locally hosted docker registry. Images can be pushed or pulled from this registry.

Create official Docker Hub account

```
$ sudo docker login
Username: username
Password:
Email: email@example.com
WARNING:login credentials saved in /home/username/.dockercfg.
Account created. Please use the confirmation link we sent to your e-mail to activate it.
```

Search publicly available images in the Docker hub registry: e.g with keyword “centos”

```
$ sudo docker search centos
```

Push customized image to Docker repository: make sure repository name meets the username of the docker hub account in order to push the images

```
$ sudo docker push username/newimage
```

Install Private Local Docker Registry

```
$ docker run -p 5000:5000 registry
```

Tag the images

Now, we will the tag the same image created in the above tutorial to “localhost:5000/newimage” so that the repository name matches and it can be easily pushed to private docker registry.

```
$docker tag username/newimage:latest localhost:5000/newimage:latest
```

Push the image to private docker registry

```
$ docker push localhost:5000/newimage:latest
```

6. Docker Networking

6.1 Bind Host Port to container Port

Run the previously created *my_ngnix* with port binding

```
$ sudo docker run -d -p 8080:80 --name test_container test/my_nginx nginx -g "daemon off;"
```

The -p `8080:80` option will bind the host `port 8080` to the container `port 80`. So we will be able to see the default web page of "Our first Docker image for Nginx" by simply visiting the IP address of our docker host.

```
$ curl http://docker-host-ip:8080
```

7. Docker Volumes

7.1 Get Started with Volumes

Let us see how to create a basic docker volume and mount it in a container

Volume mounted in a container

1. Use *-v* parameter

   ```
   $ docker run -v /volume1 -i -t ubuntu /bin/bash
   ```

2. Verify existence of volume using *ls* command

   ```
   $root@b979b5d735b0:/# ls
   bin  boot  dev  etc  home  lib  lib64..  var  volume1
   ```

Mount a Host Directory as a Volume

1. Create a local directory */host/logs*

   ```
   $ sudo mkdir -p /host/logs
   $ sudo touch /host/logs/log1.txt
   ```

2. Bind */host/logs* host directory to */container/logs* volume

   ```
   $ docker run -v /host/logs:/container/logs -i -t ubuntu  /bin/bash
   ```

3. Execute ls to see the contents of mounted volume. The contents are the same as created in host directory.

   ```
   root@3774005dee32:/# ls
   bin  boot  container  .. srv  sys  tmp  usr  var
   root@3774005dee32:/# ls container/logs/
   ```

```
log1.txt
```

8. Docker Applications

8.1 Static site on nginx server from Docker

Let's host a static site running on nginx server hosted by Docker

1. Create a Dockerfile

```
FROM ubuntu:20.04
RUN apt-get update
RUN apt-get install nginx -y
COPY index.html /var/www/html/
EXPOSE 80
CMD ["nginx", "-g", "daemon off:"]
```

2. Create a directory *public_html* with the following content in *index.html*

```
<html>
<h1>Hello World!</h1>
<p>Hi there – This is a simple static website!</p>
</html>
```

3. Your directory should look like this

```
$ tree .
.
├── Dockerfile
└── public_html
    └── index.html
```

4. Create a Docker image

```
$ docker build -t my-nginx .
```

5. Create a Docker Container running this image

```
$ docker run  -p 80:80 --name my-nginx-01  my-nginx
```

6. Open browser of the host at http://localhost:80, you will see the website up and running

Should you have any comments or recommendations regarding this guide, please drop us a note on our helpdesk at helpdesk@h3abionet.org.

--oOo END oOo--