**Docker**

**Docker is open source software to pack, ship and run any application as a lightweight container.**

**What can Docker can do for you?**

Docker solves many of the same problem that a VM solves, plus some other that VMs could solve if they didn’t were so resource intensive. Here are some of the things that Docker can deal with:

* Isolating an application dependencies
* Creating an application image and replicating it
* Creating ready to start applications that are easily distributable
* Allowing easy and fast scalation of instances
* Testing out applications and disposing them afterwards

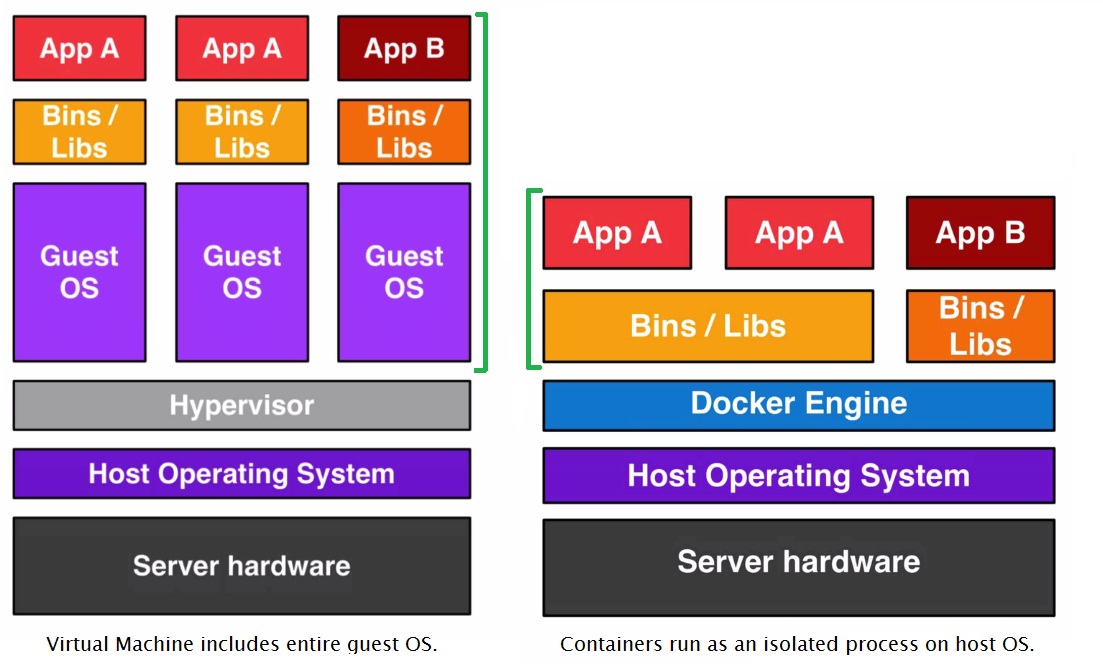
The idea behind Docker is to create **portable lightweight** **containers for software applications**that can be run on any machine with Docker installed, regardless of the underlying OS, akin to the cargo containers used on ships. Pretty ambitious, and they’re succeeding.

The two most important entities in Docker are**images**and **containers**. Aside from those, **links**and **volumes** are also important. Let’s start with images.

*Images*

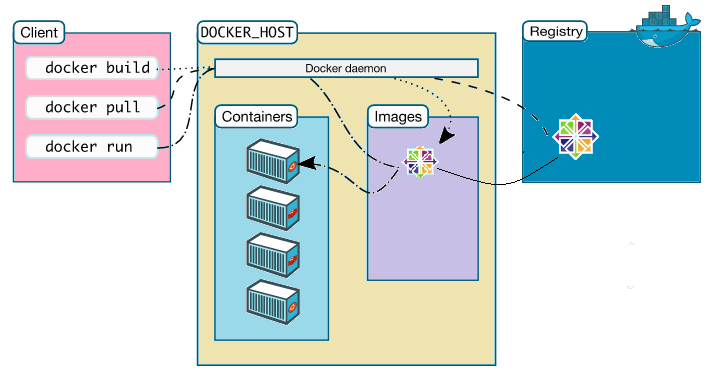
Images on Docker are like the snapshot of a virtual machine, but way more lightweight, way way way more lightweight.

A beginner often tries to correlate application container technology with build tools like Ant, Maven or Gradle. Sometimes we correlate them with configuration management tools like Chef and Puppet. They don’t fit into these categories and neither in continuous integration tool like Jenkins. They are very close to virtualization tools like VMware, but they are not exactly a virtual machine. The difference between a virtual machine and a container is better explained using a diagram shown below.

[](http://www.teckstory.com/wp-content/uploads/2015/10/Vm-vs-Container.jpg)

Docker Architecture

So far, we understand that Docker is lightweight container virtualization platform. It resembles with virtual machines, but it is not a hypervisor technology which contains a thick operating system. Application container is at the core of Docker technology and it provides a way to run almost any application securely isolated in a container. The isolation and security allow you to run many containers simultaneously on your host. The lightweight nature of containers, which run without the extra load of a hypervisor, means you can get more out of your hardware. Docker uses a client-server architecture as shown below.

[](http://www.teckstory.com/wp-content/uploads/2015/10/Docker-Architecture.png)

The Docker **client** talks to the Docker **daemon**, which does the heavy lifting of the building, running, and distributing your Docker containers. Both the Docker client and the daemon can run on the same system or you can connect to a remote Docker daemon. The user does not directly interact with the daemon, but instead through the Docker client.

A Docker **image** is a read-only template. For example, an image could contain a CentOS operating system with your application installed. Images are used to create Docker containers. you can create your own image or you can download existing images from the **registry** that other people have already created.

Docker registries hold images. These are public or private stores from which you upload or download images. The **Docker Hub**is the most popular public Docker registry.

A Docker **container** holds everything that is needed for an application to run. Each container is created from a Docker image. Docker containers can be run, started, stopped, moved, and deleted. Each container is an isolated and secure application platform.

The Docker process is summarized below.

* You can build Docker images that hold your applications.
* You can create Docker containers from those Docker images to run your applications.
* You can share those Docker images via Docker registry.

Installing Docker

There is no better method to learn something other than by doing. So let’s install Docker and start exploring it. Installing Docker is very simple. You can follow instructions from docker.com. I am using CentOS and followed instructions to [install Docker in CentOS](https://docs.docker.com/installation/centos/). Docker is available on windows, Mac and all major distributions of Linux. You can follow installation instructions specific to your distribution.

Docker Hands-on

So far, we understand that we can create containers using images. The container is the run component of Docker architecture. It holds everything that is needed for an application to run. Docker **image** is a read-only template. We can pull Docker image from the registry and use it to create a container. Let’s try it now. We will pull centos 6.6 images from the Docker hub.

Before we pull centos image, let’s check locally available images. You can list all locally available Docker images using Docker images command. In this command, docker is the client as shown in Docker architecture. The images is an option to Docker client which will list all locally available images.

[root@node1 ~]# docker images

REPOSITORY          TAG    IMAGEID  CREATED   VIRTUAL SIZE

hello-world         latest af340544ed62  8 weeks ago    960 B

I have only one local image. The hello-world image which I used to test my Docker installation. If you don’t need a local image, you can remove it. to remove a local image, we can use rmi option to Docker client and specify image name.

[root@node1 ~]# docker rmi hello-world

Untagged: hello-world:latest

Deleted: af340544ed62de0680f441c71fa1a80cb084678fed42bae393e543faea3a572c

We have already seen how to list all local image and remove them. Let’s pull centos image. You can search for available CentOS images at [Docker Hub](https://hub.docker.com/). I am going to pull CentOS 6.6 image using pull command as shown below. We need to specify image name for the pull command. An image may have multiple tags. For example, CentOS image has many tags. We can specify a tag with the image name as name: tag. Pulling a Docker image from Docker hub will take some time depending upon your internet speed.

[root@node1 ~]# docker pull centos:6.6

6.6: Pulling from library/centos

f1b10cd84249: Pull complete

8b44529354f3: Pull complete

library/centos:6.6: The image you are pulling has been verified. Important: image verification is a tech preview feature and should not be relied on to provide security.

Digest: sha256:0e69a5fa0ce3e1490d02f024a04003aa7c49e3bb320e9860ca4cc151a2aca8b1

Status: Downloaded newer image for centos:6.6

We downloaded CentOS image. it’s time to start a container using this image. To create a container from a Docker image, we use Docker run command. The general format for Docker run command is as shown below.

**docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG…]**

An example to start a Docker container is shown below. I am using four options in below example. These options are the name, h, I and t. After all options, we have Docker image name and tag which will be used as a template to start this Docker container. The last part of Docker run command is the command which I want to execute in this container. I gave a bash shell, so my container will be executing a bash shell. The -it option will give me an interactive terminal to work with the container shell.

[root@node1 ~]# docker run --name mnode -h mnode.teckstory.com -it centos:6.6 /bin/bash

The –name option gives a name to this container. If you start another terminal and issue docker ps command, it will list all active containers as shown below. The name of the container is handy while working with the container. If you don’t give a name to your container, you may have to use container id while referring your container.

[root@node1 ~]# docker ps

CONTAINERID    IMAGE      COMMAND   CREATED  STATUS  PORTS  NAMES

60500b452162  centos:6.6 "/bin/bash" 3 minutes ago  Up3minutes  mnode

The -h option assigns a hostname to this container. Once you container is started, You can check hostname for the container as shown below.

[root@mnode /]# hostname -f

mnode.teckstory.com

All Docker containers by default come with network capability. My container is assigned an IP address.

[root@mnode /]# ifconfig

eth0      Link encap:Ethernet  HWaddr 02:42:AC:11:00:05

inet addr:172.17.0.5  Bcast:0.0.0.0  Mask:255.255.0.0

inet6 addr: fe80::42:acff:fe11:5/64 Scope:Link

Ok, What if I ping it from another terminal?

[root@node1 ~]# ping 172.17.0.5

PING 172.17.0.5 (172.17.0.5) 56(84) bytes of data.

64 bytes from 172.17.0.5: icmp\_seq=1 ttl=64 time=0.515 ms

64 bytes from 172.17.0.5: icmp\_seq=2 ttl=64 time=0.134 ms

64 bytes from 172.17.0.5: icmp\_seq=3 ttl=64 time=0.137 ms

64 bytes from 172.17.0.5: icmp\_seq=4 ttl=64 time=0.133 ms

This is great, I have CentOS 6.6 container (a kind of VM) running within my host. Let’s take it further and start another container.

[root@node1 ~]# docker run --name data1 -h data1.teckstory.com -it centos:6.6 /bin/bash

The Hostname for this new container.

[root@data1 /]# hostname -f

data1.teckstory.com

The IP for the new container.

[root@data1 /]# ifconfig

eth0      Link encap:Ethernet  HWaddr 02:42:AC:11:00:06

inet addr:172.17.0.6  Bcast:0.0.0.0  Mask:255.255.0.0

Can I ping the first container from this container?

[root@data1 /]# ping 172.17.0.5

PING 172.17.0.5 (172.17.0.5) 56(84) bytes of data.

64 bytes from 172.17.0.5: icmp\_seq=1 ttl=64 time=0.245 ms

64 bytes from 172.17.0.5: icmp\_seq=2 ttl=64 time=0.133 ms

Can I ssh from one container to another container or from my host to a container? Yes, I can if I have ssh service installed and running. Let’s try it.

[root@data1 /]# yum install openssh-server openssh-clients

............

Installed:

openssh-clients.x86\_64 0:5.3p1-112.el6\_7

openssh-server.x86\_64 0:5.3p1-112.el6\_7

Dependency Installed:

fipscheck.x86\_64 0:1.2.0-7.el6

fipscheck-lib.x86\_64 0:1.2.0-7.el6

libedit.x86\_64 0:2.11-4.20080712cvs.1.el6

openssh.x86\_64 0:5.3p1-112.el6\_7

tcp\_wrappers-libs.x86\_64 0:7.6-57.el6

Complete!

Let’s start ssh service and create a user which I will use to login to this container.

[root@data1 /]# service sshd start

Generating SSH2 RSA host key:                              [  OK  ]

Generating SSH1 RSA host key:                              [  OK  ]

Generating SSH2 DSA host key:                              [  OK  ]

Starting sshd:                                             [  OK  ]

[root@data1 /]# chkconfig sshd on

[root@data1 /]# useradd pkp

[root@data1 /]# passwd pkp

Changing password for user pkp.

New password:

BAD PASSWORD: it is based on a dictionary word

BAD PASSWORD: is too simple

Retype new password:

passwd: all authentication tokens updated successfully.

Let’s try ssh from another container. Before I can do it, I have to install OpenSSH-clients.

[root@mnode /]# ssh pkp@172.17.0.6

The authenticity of host '172.17.0.6 (172.17.0.6)' can't be established.

RSA key fingerprint is 9d:33:21:35:5c:e8:07:8b:c8:6f:a8:21:bd:4f:95:9e.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added '172.17.0.6' (RSA) to the list of known hosts.

pkp@172.17.0.6's password:

[pkp@data1 ~]$

Can I ssh from host machine?

[root@node1 ~]# ssh pkp@172.17.0.6

The authenticity of host '172.17.0.6 (172.17.0.6)' can't be established.

RSA key fingerprint is 9d:33:21:35:5c:e8:07:8b:c8:6f:a8:21:bd:4f:95:9e.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added '172.17.0.6' (RSA) to the list of known hosts.

pkp@172.17.0.6's password:

Last login: Sat Oct  3 16:21:33 2015 from mnode

[pkp@data1 ~]$

Let’s stop both containers.

[root@node1 ~]# docker stop mnode

[root@node1 ~]# docker stop data1

[root@node1 ~]# docker ps -a

CONTAINERID  IMAGE   COMMAND  CREATED  STATUS  PORTS   NAMES

10fc91706f99 centos:6.6 "/bin/bash" 36 minutes ago Exited (137)30seconds ago data1

60500b452162 centos:6.6 "/bin/bash" 47 minutes ago Exited (137) 50 seconds ago   mnode

The docker ps -a will list all containers even if they are stopped.