**[Docker image and container via docker commands (search, pull, run, ps, restart, attach, and rm)](http://www.bogotobogo.com/DevOps/Docker/Docker_Commands_for_Images_Container.php)**

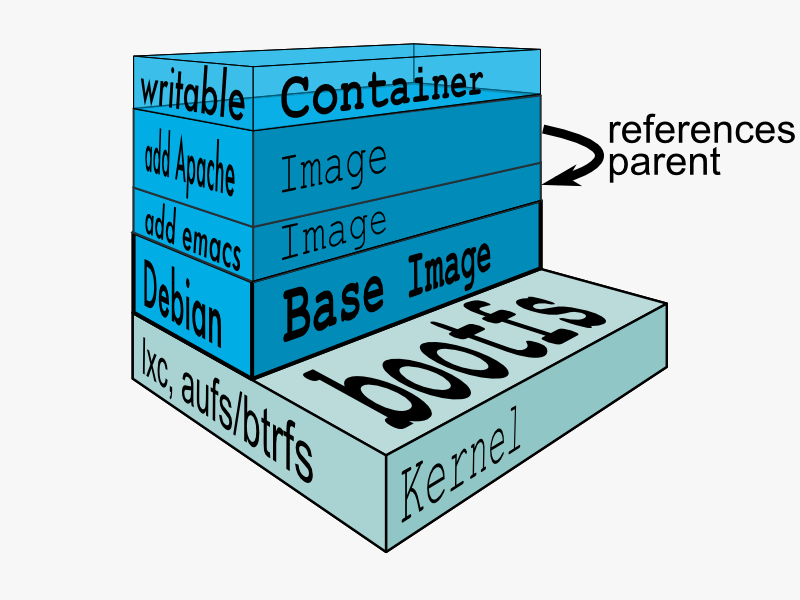
**LinuX Containers (LXC)**

The problem with Virtual Machines built using VirtualBox or VMWare is that we have to run entire OS for every VM. That's where Docker comes in. Docker virtualizes on top of one OS so that we can run Linux using technology known as LinuX Containers (LXC). LXC combines cgroups and namespace support to provide an isolated environment for applications. Docker can also use LXC as one of its execution drivers, enabling image management and providing deployment services.

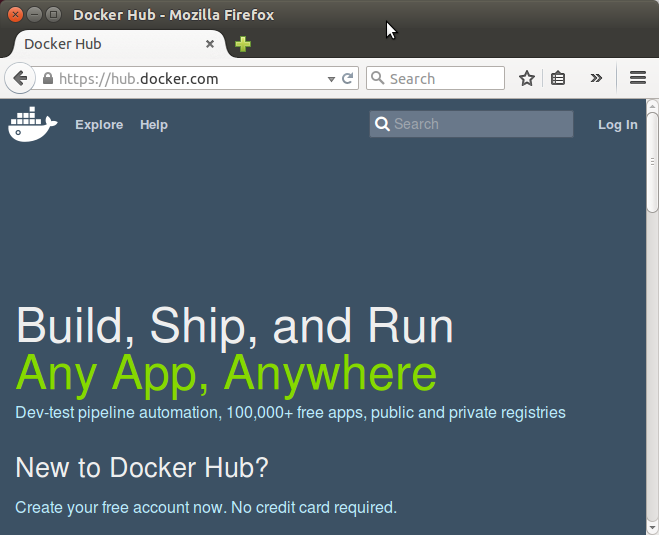
Docker allows us to run applications inside containers. Running an application inside a container takes a single command: **docker run**.

**Docker Registry - Repositories of Docker Images**

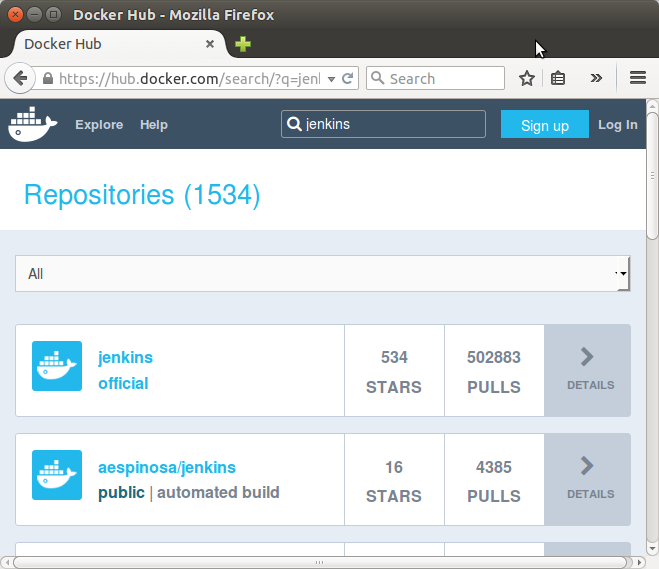
We need to have a **disk image** to make the virtualization work. The disk image represents the system we're running on and they are the basis of containers.



Docker registry is a registry of already existing images that we can use to run and create containerized applications.



As we can see from the picture above, there are lots of communities and works already been done to build the system. Docker company supports and maintains its registry and the community around it.



We can search images within the registry hub, for example, the sample picture is the result from searching "Jenkins".

**Docker search**

The Docker search command allows us to go and look at the registry in search for the images that we want.

$ docker search --help

Usage: docker search [OPTIONS] TERM

Search the Docker Hub for images

--automated=false Only show automated builds

--no-trunc=false Don't truncate output

-s, --stars=0 Only displays with at least x stars

If we do the same search, Jenkins, we get exactly the same result as we got from the web:

$ docker search jenkins

NAME DESCRIPTION STARS OFFICIAL AUTOMATED

jenkins Official Jenkins Docker image 158 [OK]

orchardup/jenkins 18 [OK]

aespinosa/jenkins Sets up a container with jenkins installed... 12 [OK]

evarga/jenkins-slave This is a basic container to be used as a ... 6 [OK]

...

rapidftr/jenkins-slave 0 [OK]

We got too many outputs, so we need to filter it out items with more than 10 stars:

$ docker search -s 10 jenkins

NAME DESCRIPTION STARS OFFICIAL AUTOMATED

jenkins Official Jenkins Docker image 158 [OK]

orchardup/jenkins 18 [OK]

aespinosa/jenkins Sets up a container with jenkins installed... 12 [OK]

**Docker pull**

Once we found the image we like to use it, we can use Docker's pull command:

$ docker pull --help

Usage: docker pull [OPTIONS] NAME[:TAG]

Pull an image or a repository from the registry

-a, --all-tags=false Download all tagged images in the repository

The pull command will go up to the web site and grab the image and download it to our local machine.

$ docker pull ubuntu

ubuntu:latest: The image you are pulling has been verified

511136ea3c5a: Already exists

d497ad3926c8: Already exists

ccb62158e970: Already exists

e791be0477f2: Already exists

3680052c0f5c: Already exists

22093c35d77b: Already exists

5506de2b643b: Already exists

Status: Image is up to date for ubuntu:latest

The pull command without any tag will download all Ubuntu images though I've already done it. To see what Docker images are available on our machine, we use docker images:

$ docker images

REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE

ubuntu latest 5506de2b643b 4 weeks ago 199.3 MB

So, the output indicates only one image is currently on my local machine. We also see the image has a TAG inside of it.

As we can see from the command below, docker pull centos:latest, we can also be more specific, and download only the version we need. In Docker, versions are marked with tags.

$ docker pull centos:latest

centos:latest: The image you are pulling has been verified

5b12ef8fd570: Pull complete

ae0c2d0bdc10: Pull complete

511136ea3c5a: Already exists

Status: Downloaded newer image for centos:latest

Here is the images on our local machine.

$ docker images

REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE

centos latest ae0c2d0bdc10 2 weeks ago 224 MB

ubuntu latest 5506de2b643b 4 weeks ago 199.3 MB

The command, **docker images**, returns the following columns:

1. **REPOSITORY**: The name of the repository, which in this case is "ubuntu".
2. **TAG**: Tags represent a specific set point in the repositories' commit history. As we can see from the list, we've pulled down different versions of linux. Each of these versions is tagged with a version number, a name, and there's even a special tag called "latest" which represents the latest version.
3. **IMAGE ID**: This is like the primary key for the image. Sometimes, such as when we commit a container without specifying a name or tag, the repository or the tag is <NONE>, but we can always refer to a specific image or container using its ID.
4. **CREATED**: The date the repository was created, as opposed to when it was pulled. This can help us assess how "fresh" a particular build is. Docker appears to update their master images on a fairly frequent basis.
5. **VIRTUAL SIZE**: The size of the image.

**Docker run**

Now we have images on our local machine. What do we do with them? This is where docker run command comes in.

$ docker run --help

Usage: docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

Run a command in a new container

-a, --attach=[] Attach to STDIN, STDOUT or STDERR.

--add-host=[] Add a custom host-to-IP mapping (host:ip)

-c, --cpu-shares=0 CPU shares (relative weight)

--cap-add=[] Add Linux capabilities

--cap-drop=[] Drop Linux capabilities

--cidfile="" Write the container ID to the file

--cpuset="" CPUs in which to allow execution (0-3, 0,1)

-d, --detach=false Detached mode: run the container in the background and print the new container ID

--device=[] Add a host device to the container (e.g. --device=/dev/sdc:/dev/xvdc)

--dns=[] Set custom DNS servers

--dns-search=[] Set custom DNS search domains

-e, --env=[] Set environment variables

--entrypoint="" Overwrite the default ENTRYPOINT of the image

--env-file=[] Read in a line delimited file of environment variables

--expose=[] Expose a port from the container without publishing it to your host

-h, --hostname="" Container host name

-i, --interactive=false Keep STDIN open even if not attached

--link=[] Add link to another container in the form of name:alias

--lxc-conf=[] (lxc exec-driver only) Add custom lxc options --lxc-conf="lxc.cgroup.cpuset.cpus = 0,1"

-m, --memory="" Memory limit (format: , where unit = b, k, m or g)

--name="" Assign a name to the container

--net="bridge" Set the Network mode for the container

'bridge': creates a new network stack for the container on the docker bridge

'none': no networking for this container

'container:': reuses another container network stack

'host': use the host network stack inside the container. Note: the host mode gives the container full access to local system services such as D-bus and is therefore considered insecure.

-P, --publish-all=false Publish all exposed ports to the host interfaces

-p, --publish=[] Publish a container's port to the host

format: ip:hostPort:containerPort | ip::containerPort | hostPort:containerPort | containerPort

(use 'docker port' to see the actual mapping)

docker run --help is a rather big help, and we have more:

--privileged=false Give extended privileges to this container

--restart="" Restart policy to apply when a container exits (no, on-failure[:max-retry], always)

--rm=false Automatically remove the container when it exits (incompatible with -d)

--security-opt=[] Security Options

--sig-proxy=true Proxy received signals to the process (even in non-TTY mode). SIGCHLD, SIGSTOP, and SIGKILL are not proxied.

-t, --tty=false Allocate a pseudo-TTY

-u, --user="" Username or UID

-v, --volume=[] Bind mount a volume (e.g., from the host: -v /host:/container, from Docker: -v /container)

--volumes-from=[] Mount volumes from the specified container(s)

-w, --workdir="" Working directory inside the container

Currently we are on Ubuntu 14.04.1 LTS machine (local):

$ cat /etc/issue

Ubuntu 14.04.1 LTS \n \l

Now we're going to Docker run centos image. This will create container based upon the image and execute the bin/bash command. Then it will take us into a shell on that machine that can continue to do things:

$ docker run -it centos:latest /bin/bash

[root@98f52715ecfa /]#

By executing it, we're now on a bash. If we look at /etc/redhat-release:

[root@98f52715ecfa /]# cat /etc/redhat-release

CentOS Linux release 7.0.1406 (Core)

We're now on CentOS 7.0 on top of my Ubuntu 14.04 machine. We have an access to yum:

[root@98f52715ecfa /]# yum

Loaded plugins: fastestmirror

You need to give some command

Usage: yum [options] COMMAND

List of Commands:

check Check for problems in the rpmdb

check-update Check for available package updates

...

Let's make a new file in our home directory:

[root@98f52715ecfa /]# ls

bin dev etc home lib lib64 lost+found media mnt opt proc root run sbin selinux srv sys tmp usr var

[root@98f52715ecfa /]# cd /home

[root@98f52715ecfa home]# ls

[root@98f52715ecfa home]# touch bogotobogo.txt

[root@98f52715ecfa home]# ls

bogotobogo.txt

[root@98f52715ecfa home]# exit

exit

k@laptop:~$

**Docker ps - list containers**

After making a new file on our Docker container, we exited from there, and we're back to our local machine with Ubuntu system.

$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

The docker ps lists containers but currently we do not have any. That's because nothing is running. It shows only running containers.

We can list all containers using -a option:

$ docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

98f52715ecfa centos:latest "/bin/bash" 12 minutes ago Exited (0) 5 minutes ago goofy\_yonath

f8c5951db6f5 ubuntu:latest "/bin/bash" 4 hours ago Exited (0) 4 hours ago furious\_almeida

**Docker restart**

How can we restart Docker container?

$ docker restart --help

Usage: docker restart [OPTIONS] CONTAINER [CONTAINER...]

Restart a running container

-t, --time=10 Number of seconds to try to stop for before killing the container. Once killed it will then be restarted. Default is 10 seconds.

We can restart the container that's already created:

$ docker restart 98f52715ecfa

98f52715ecfa

k@laptop:~$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

98f52715ecfa centos:latest "/bin/bash" 20 minutes ago Up 10 seconds goofy\_yonath

Now we have one active running container, and it already executed the /bin/bashcommand.

**Docker attach**

The **docker attach** command allows us to attach to a running container using the container's ID or name, either to view its ongoing output or to control it interactively. We can attach to the same contained process multiple times simultaneously, screen sharing style, or quickly view the progress of our daemonized process.

$ docker attach --help

Usage: docker attach [OPTIONS] CONTAINER

Attach to a running container

--no-stdin=false Do not attach STDIN

--sig-proxy=true Proxy all received signals to the process (even in non-TTY mode). SIGCHLD, SIGKILL, and SIGSTOP are not proxied.

We can attach to a running container:

$ docker attach 98f52715ecfa

[root@98f52715ecfa /]#

[root@98f52715ecfa /]# cd /home

[root@98f52715ecfa home]# ls

bogotobogo.txt

Now we're back to the CentOS container we've created, and the file we made is still there in our home directory.

**Docker rm**

We can delete the container:

[root@98f52715ecfa home]# exit

exit

k@laptop:~$ docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

98f52715ecfa centos:latest "/bin/bash" 30 minutes ago Exited (0) 14 seconds ago goofy\_yonath

f8c5951db6f5 ubuntu:latest "/bin/bash" 5 hours ago Exited (0) 5 hours ago furious\_almeida

k@laptop:~$ docker rm f8c5951db6f5

f8c5951db6f5

k@laptop:~$ docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

98f52715ecfa centos:latest "/bin/bash" 32 minutes ago Exited (0) 2 minutes ago goofy\_yonath

We deleted the Ubuntu container and now we have only one container, CentOS.

**Remove all images and containers**

We use Docker, but working with it creates lots of images and containers. So, we may want to remove all of them to save disk space.

To delete all containers:

$ docker rm $(docker ps -a -q)

To delete all images:

$ docker rmi $(docker images -q)

Here the **-a** and **-q** do this:

1. **-a**: Show all containers (default shows just running)
2. **-q**: Only display numeric IDs