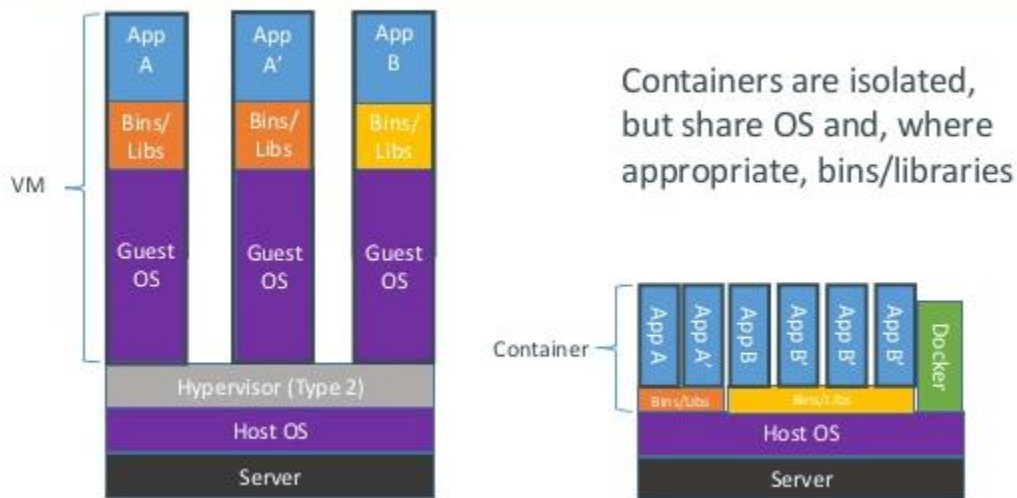


## Containers vs. VMs



### Docker commands:

- Want to check docker version use below command.  
**\$ docker -v (or) \$ docker --version**
- Check complete docker information use below commands.  
**\$ docker info**
- Check what are all the option can we use in docker use below command.  
**\$ docker**
- List out the docker images use below command.  
**\$ docker images**
- If you want to search any docker images use below command.  
**\$ docker search [option]**  
**Ex: docker search Ubuntu**
- If you want to run docker images use below command.  
**Note:** if don't have image locally its download first and then run automatically.  
**\$ docker run --name <container-name> <image-name>**  
**\$ docker run -t --name <container-name> <image-name>**  
**\$ docker run -d -p 8181:8080 --name myapp pavithriya/webapp:v1**
- If we want to download any docker image from docker hub or docker registry we need to use below command.  
**\$ docker pull <image name>**  
**Ex: docker pull tutum/hello-world**

**\$ docker run -p 80 --name hello tutum/hello-world**

**Note:** if we want check outside access port on the running container port use below command.

**\$ sudo docker port d35bf1374e88 80**

**\$ docker run -p 1234:80 --name hello tutum/hello-world**

**Note:** if we didn't mention label (version) it will download latest version of the image.

8. if we want to check the running docker containers use below command.

**\$ docker ps**

9. if we want to check running and stopping containers use below command.

**\$ docker ps -a**

10. if want to remove the container name use below command.

**\$ docker rm <containername>**

11. remove the docker images from the list.

**\$ docker rmi <image-name>**

12. check only docker container ID's use below command.

**\$ docker ps -q**

13. to check latest running docker container use below command.

**\$ docker ps -l**

14. to kill running docker container use below command.

**\$ docker kill <container id>**

15. to stop the running container use below command.

**\$ docker stop <container id>**

**Diff b/w kill and stop :** Both will stop the running container difference is stop will take 10 sec of time to stop the container(first it will send the signal then its stopped )kill won't send any signal it will stop immediately.

16. To run the stopped container use below command.

**\$ docker ps -a**

**\$ docker start <container id>**

17. If we want to check the logs of back round running container use below command.

**\$ docker ps -a**

**\$ docker logs <container id>**

**\$ docker logs 35nfdw453** (or use first three characters of container id also ex: **\$ docker logs 35n** )

If we want to check the follow up logs of running container use below command

**\$ docker logs -f -t 1 <container ID>**

18. If want to login any docker registry use below command. For example I want to login docker hub(it's a public repository ).

**\$ docker login**

Enter username and password of docker hub.

19. Build a custom docker image use below command.

```
$docker build -it ybmsr/hello:v1 .
```

Without cache

```
$dcoker build - -no-cache -t ybmsr/hello:v1 .
```

20. Push docker image to docker hub or any repository use below command.

```
$ docker push ybmsr/hello:v1
```

21. To check running docker container information use below command.

```
$ docker inspect <container ID>
```

## Docker Tutorial

### Introduction

Docker is an application that makes it simple and easy to run application processes in a container, which are like virtual machines, only more portable, more resource-friendly, and more dependent on the host operating system.

### Installing Docker

The Docker installation package available in the official Ubuntu 16.04 repository may not be the latest version. To get this latest version, install Docker from the official Docker repository. This section shows you how to do just that.

First, in order to ensure the downloads are valid, 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 -
```

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-get update
```

Make sure you are about to install from the Docker repo instead of the default Ubuntu 16.04 repo:

```
$ apt-cache policy docker-ce
```

You should see output similar to the follow:

```
root@vuser-HVM-domU:~# apt-cache policy docker-ce
docker-ce:
  Installed: (none)
  Candidate: 18.06.1~ce~3-0~ubuntu
  Version table:
   18.06.1~ce~3-0~ubuntu 500
      500 https://download.docker.com/linux/ubuntu xenial/stable amd64 Packages
   18.06.0~ce~3-0~ubuntu 500
      500 https://download.docker.com/linux/ubuntu xenial/stable amd64 Packages
   18.03.1~ce~0~ubuntu 500
      500 https://download.docker.com/linux/ubuntu xenial/stable amd64 Packages
   18.03.0~ce~0~ubuntu 500
      500 https://download.docker.com/linux/ubuntu xenial/stable amd64 Packages
   17.12.1~ce~0~ubuntu 500
      500 https://download.docker.com/linux/ubuntu xenial/stable amd64 Packages
```

Notice that *docker-ce* is not installed, but the candidate for installation is from the Docker repository for Ubuntu 16.04 (*xenial*).

Finally, install Docker:

```
$ sudo apt-get install -y docker-ce
```

Docker should now be installed, the daemon started, and the process enabled to start on boot. Check that it's running:

```
$ sudo systemctl status docker
```

The output should be similar to the following, showing that the service is active and running:

```
root@vuser-HVM-domU:~# sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2018-10-22 13:54:02 IST; 8min ago
     Docs: https://docs.docker.com
   Main PID: 26502 (dockerd)
    Tasks: 24
   Memory: 76.4M
      CPU: 1.681s
   CGroup: /system.slice/docker.service
           └─26502 /usr/bin/dockerd -H fd://
           └─26535 docker-containerd --config /var/run/docker/containerd/containerd.toml
```

Installing Docker now gives you not just the Docker service (daemon) but also the *docker* command line utility, or the Docker client. We'll explore how to use the *docker* command later in this tutorial.

## Executing the Docker Command Without Sudo

By default, running the docker command requires *root* privileges — that is, you have to prefix the command with *sudo*. It can also be run by a user in the **docker** group, which is automatically created during the installation of Docker. If you attempt to run the *docker* command without prefixing it with *sudo* or without being in the docker group, you'll get an output like this:

```
Output
docker: Cannot connect to the Docker daemon. Is the docker daemon running on
this host?.
See 'docker run --help'.
```

If you want to avoid typing *sudo* whenever you run the *docker* command, add your username to the *docker* group:

```
$ sudo usermod -aG docker ${USER}
```

To apply the new group membership, you can log out of the server and back in, or you can type the following:

```
$ su - ${USER}
```

You will be prompted to enter your user's password to continue. Afterwards, you can confirm that your user is now added to the *docker* group by typing:

```
$ id -nG
```

If you need to add a user to the *docker* group that you're not logged in as, declare that username explicitly using:

```
$ sudo usermod -aG docker <username>
```

## Using the Docker Command

With *Docker* installed and working, now's the time to become familiar with the command line utility. Using *docker* consists of passing it a chain of options and commands followed by arguments. The syntax takes this form:

```
$ docker [option] [command] [arguments]
```

To view all available subcommands, type:

```
$ docker
```

As of Docker 18.06.1, the complete list of available subcommands includes:

```
root@vuser-HVM-domU:~# docker

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
  --config string      Location of client config files (default "/root/.docker")
  -D, --debug          Enable debug mode
  -H, --host list      Daemon socket(s) to connect to
  -l, --log-level string Set the logging level ("debug"|"info"|"warn"|"error"|"fatal") (default "info")
  --tls               Use TLS; implied by --tlsverify
  --tlscacert string  Trust certs signed only by this CA (default "/root/.docker/ca.pem")
  --tlscert string    Path to TLS certificate file (default "/root/.docker/cert.pem")
  --tlskey string     Path to TLS key file (default "/root/.docker/key.pem")
  --tlsverify         Use TLS and verify the remote
  -v, --version       Print version information and quit

Management Commands:
  config      Manage Docker configs
  container   Manage containers
  image       Manage images
  network     Manage networks
  node        Manage Swarm nodes
  plugin      Manage plugins
  secret      Manage Docker secrets
  service     Manage services
  stack       Manage Docker stacks
  swarm       Manage Swarm
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes

Commands:
```

```
attach      Attach local standard input, output, and error streams to a running container
build       Build an image from a Dockerfile
commit      Create a new image from a container's changes
cp          Copy files/folders between a container and the local filesystem
create      Create a new container
diff        Inspect changes to files or directories on a container's filesystem
events      Get real time events from the server
exec        Run a command in a running container
export      Export a container's filesystem as a tar archive
history     Show the history of an image
images      List images
import      Import the contents from a tarball to create a filesystem image
info        Display system-wide information
inspect     Return low-level information on Docker objects
kill        Kill one or more running containers
load        Load an image from a tar archive or STDIN
login       Log in to a Docker registry
logout      Log out from a Docker registry
logs        Fetch the logs of a container
pause       Pause all processes within one or more containers
port        List port mappings or a specific mapping for the container
ps          List containers
pull        Pull an image or a repository from a registry
push        Push an image or a repository to a registry
rename      Rename a container
restart     Restart one or more containers
rm          Remove one or more containers
rmi         Remove one or more images
run         Run a command in a new container
save        Save one or more images to a tar archive (streamed to STDOUT by default)
search      Search the Docker Hub for images
start       Start one or more stopped containers
stats       Display a live stream of container(s) resource usage statistics
stop        Stop one or more running containers
tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
top         Display the running processes of a container
unpause     Unpause all processes within one or more containers
update      Update configuration of one or more containers
version     Show the Docker version information
wait        Block until one or more containers stop, then print their exit codes

Run 'docker COMMAND --help' for more information on a command.
```

To view system-wide information about Docker, use:

```
$ docker info
```

```
root@vuser-HVM-domU:~# docker info
Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
Images: 0
Server Version: 18.06.1-ce
Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
  Volume: local
  Network: bridge host macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file logentries splunk syslog
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 468a545b9edcd5932818eb9de8e72413e616e86e
runc version: 69663f0bd4b60df09991c08812a60108003fa340
init version: fec3683
Security Options:
  apparmor
  seccomp
   Profile: default
Kernel Version: 4.4.0-47-generic
Operating System: Ubuntu 16.04 LTS
OSType: linux
Architecture: x86_64
CPUs: 2
Total Memory: 9.757GiB
Name: vuser-HVM-domU
ID: KCCN:OS6A:QTKY:E3BU:WFLG:YTAT:CSBF:S4UR:7S3E:VZJR:VWWE:J37I
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://index.docker.io/v1/
Labels:
Experimental: false
Insecure Registries:
  127.0.0.0/8
Live Restore Enabled: false

WARNING: No swap limit support
root@vuser-HVM-domU:~#
```



## Working with Docker Images

Docker containers are run from Docker images. By default, it pulls these images from Docker Hub, a Docker registry managed by Docker, the company behind the Docker project. Anybody can build and host their Docker images on Docker Hub, so most applications and Linux distributions you'll need to run Docker containers have images that are hosted on Docker Hub.

To check whether you can access and download images from Docker Hub, type:

```
$ docker run hello-world
```

In the output, you should see the following message, which indicates that Docker is working correctly:

```
root@vuser-HVM-domU:~# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
d1725b59e92d: Pull complete
Digest: sha256:0add3ace90ecb4adbf777e9aacf18357296e799f81cab9fde470971e499788
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

You can search for images available on Docker Hub by using the *docker* command with the *search* subcommand. For example, to search for the Ubuntu image, type:

```
$ docker search ubuntu
```

The script will crawl Docker Hub and return a listing of all images whose name matches the search string. In this case, the output will be similar to this:

```
root@vuser-HVM-domU:~# docker search ubuntu
```

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
ubuntu	Ubuntu is a Debian-based Linux operating sys...	8580	[OK]	
dorowu/ubuntu-desktop-lxde-vnc	Ubuntu with openssh-server and NoVNC	230		[OK]
rastasheep/ubuntu-sshd	Dockerized SSH service, built on top of offi...	176		[OK]
consol/ubuntu-xfce-vnc	Ubuntu container with "headless" VNC session...	129		[OK]
ansible/ubuntu14.04-ansible	Ubuntu 14.04 LTS with ansible	95		[OK]
ubuntu-upstart	Upstart is an event-based replacement for th...	92	[OK]	
neurodebian	NeuroDebian provides neuroscience research s...	54	[OK]	
landinternet/ubuntu-16-nginx-php-phpmyadmin-mysql-5	ubuntu-16-nginx-php-phpmyadmin-mysql-5	48		[OK]
ubuntu-debootstrap	debootstrap --variant=minbase --components=m...	39	[OK]	
nuagebec/ubuntu	Simple always updated Ubuntu docker images w...	23		[OK]
tutum/ubuntu	Simple Ubuntu docker images with SSH access	18		[OK]
i386/ubuntu	Ubuntu is a Debian-based Linux operating sys...	14		
landinternet/ubuntu-16-apache-php-7.0	ubuntu-16-apache-php-7.0	13		[OK]
ppc64le/ubuntu	Ubuntu is a Debian-based Linux operating sys...	12		
eclipse/ubuntu_jdk8	Ubuntu, JDK8, Maven 3, git, curl, nmap, mc, ...	6		[OK]
landinternet/ubuntu-16-nginx-php-5.6-wordpress-4	ubuntu-16-nginx-php-5.6-wordpress-4	6		[OK]
codenvy/ubuntu_jdk8	Ubuntu, JDK8, Maven 3, git, curl, nmap, mc, ...	4		[OK]
darksheer/ubuntu	Base Ubuntu Image -- Updated hourly	4		[OK]
pivotaldata/ubuntu	A quick freshening-up of the base Ubuntu doc...	2		

In the **OFFICIAL** column, **OK** indicates an image built and supported by the company behind the project. Once you've identified the image that you would like to use, you can download it to your computer using the pull subcommand. Try this with the ubuntu image, like so:

```
$ docker pull ubuntu
```

```
root@vuser-HVM-domU:~# docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
473ede7ed136: Pull complete
c46b5fa4d940: Pull complete
93ae3df89c92: Pull complete
6b1eed27cade: Pull complete
Digest: sha256:29934af957c53004d7fb6340139880d23fb1952505a15d69a03af0d1418878cb
Status: Downloaded newer image for ubuntu:latest
root@vuser-HVM-domU:~#
```

After an image has been downloaded, you may then run a container using the downloaded image with the run subcommand. If an image has not been downloaded when docker is executed with the run subcommand, the Docker client will first download the image, then run a container using it:

```
$ docker run ubuntu
```

To see the images that have been downloaded to your computer, type:

```
$ docker images
```

The output should look similar to the following:

```
root@vuser-HVM-domU:~# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	latest	ea4c82dcd15a	3 days ago	85.8MB
hello-world	latest	4ab4c602aa5e	6 weeks ago	1.84kB

## Running a Docker Container

The **hello-world** container you ran in the previous step is an example of a container that runs and exits after emitting a test message. Containers can be much more useful than that, and they can be interactive. After all, they are similar to virtual machines, only more resource-friendly.

As an example, let's run a container using the latest image of Ubuntu. The combination of the **-i** and **-t** switches gives you interactive shell access into the container:

```
$ docker run -it ubuntu
```

**Note:** The default behavior for the run command is to start a new container. Once you run the preceding command, you will open up the shell interface of a second ubuntu container.

```
root@vuser-HVM-domU:~# docker run -it ubuntu
root@1e1fddfa2dd8:/#
```

Your command prompt should change to reflect the fact that you're now working inside the container and should take this form:

Output

```
$ root@9b0db8a30ad1:/#
```

**Note:** Remember the container id in the command prompt. In the preceding example, it is **9b0db8a30ad1**. You'll need that container ID later to identify the container when you want to remove it.

Now you can run any command inside the container. For example, let's update the package database inside the container. You don't need to prefix any command with **sudo**, because you're operating inside the container as the **root** user:

```
$ root@9b0db8a30ad1:/# apt-get update
```

Then install any application in it. Let's install Node.js:

```
$ root@9b0db8a30ad1:/# apt-get install -y nodejs
```

This installs Node.js in the container from the official Ubuntu repository. When the installation finishes, verify that Node.js is installed:

```
$ root@9b0db8a30ad1:/# node -v
```

You'll see the version number displayed in your terminal:

```
root@1e1fddfa2dd8:/# node -v
v8.10.0
root@1e1fddfa2dd8:/#
```

Any changes you make inside the container only apply to that container.

To exit the container, type exit at the prompt.

## Managing Docker Containers

After using Docker for a while, you'll have many active (running) and inactive containers on your computer. To view the **active ones**, use:

```
$ docker ps
```

You will see output similar to the following:

```
root@vuser-HVM-domU:~# docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS        NAMES
root@vuser-HVM-domU:~#
```

In this tutorial, you started three containers; one from the **hello-world** image and two from the **ubuntu** image. These containers are no longer running, but they still exist on your system.

To view all containers — **active and inactive** — run **docker ps** with the **-a** switch:

```
$ docker ps -a
```

```
root@vuser-HVM-domU:~# docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS        NAMES
1e1fddfa2dd8   ubuntu    "/bin/bash"             36 minutes ago Exited (0) 2 minutes ago                practical_bartik
42d7627182bd   ubuntu    "/bin/bash"             About an hour ago Exited (0) About an hour ago                wonderful_jepsen
502f83876230   hello-world "/hello"                 About an hour ago Exited (0) About an hour ago                zen_chebyshev
```

To view the latest container you created, pass it the **-l** switch:

```
$ docker ps -l
```

```
root@vuser-HVM-domU:~# docker ps -l
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS        NAMES
1e1fddfa2dd8   ubuntu    "/bin/bash"             About an hour ago Exited (0) About an hour ago                practical_bartik
```

To start a stopped container, use **docker start**, followed by the **container ID** or the container's name. Let's start the Ubuntu-based container with the ID of **9b0db8a30ad1**:

```
$ docker start 1e1fddfa2dd8
```

```
root@vuser-HVM-domU:~# docker start 1e1fddfa2dd8
1e1fddfa2dd8
```

The container will start, and you can use **docker ps** to see its status:

```
$ docker ps
```

```
root@vuser-HVM-domU:~# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
1e1fddfa2dd8	ubuntu	"/bin/bash"	2 hours ago	Up About a minute		practical_bartik

To stop a running container, use **docker stop**, followed by the **container ID** or **name**. This time, we'll use the name that Docker assigned the container, which is *practical\_bartik*:

```
$ docker stop practical_bartik
```

```
root@vuser-HVM-domU:~# docker stop practical_bartik
practical_bartik
```

Once you've decided you no longer need a container anymore, remove it with the **docker rm** command, again using either the **container ID** or the **name**. Use the **docker ps -a** command to find the container ID or name for the container associated with the **hello-world** image and remove it.

```
$ docker rm 502f83876230
```

```
root@vuser-HVM-domU:~# docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
1e1fddfa2dd8	ubuntu	"/bin/bash"	2 hours ago	Exited (0) 5 minutes ago		practical_bartik
42d7627182bd	ubuntu	"/bin/bash"	2 hours ago	Exited (0) 2 hours ago		wonderful_jepsen
502f83876230	hello-world	"/hello"	3 hours ago	Exited (0) 3 hours ago		zen_chebyshev

```
root@vuser-HVM-domU:~# docker rm 502f83876230
502f83876230
root@vuser-HVM-domU:~#
```

You can start a new container and give it a name using the **--name** switch. You can also use the **--rm** switch to create a container that removes itself when it's stopped.

```
root@vuser-HVM-domU:~# docker run --name myubuntu ubuntu
```

```
root@vuser-HVM-domU:~# docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
6b6b363079ee	ubuntu	"/bin/bash"	21 seconds ago	Exited (0) 19 seconds ago		myubuntu
1e1fddfa2dd8	ubuntu	"/bin/bash"	2 hours ago	Exited (0) 23 minutes ago		practical_bartik
42d7627182bd	ubuntu	"/bin/bash"	3 hours ago	Exited (0) 3 hours ago		wonderful_jepsen

If we use **-rm** option along with **--name** once stop the container it will delete automatically. See below *myubuntu\_new* container is not showing.

```
root@vuser-HVM-domU:~# docker run --rm --name myubuntu_new ubuntu
root@vuser-HVM-domU:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
6b6b363079ee        ubuntu             "/bin/bash"        2 minutes ago      Exited (0) 2 minutes ago              myubuntu
1e1fdafa2dd8        ubuntu             "/bin/bash"        2 hours ago        Exited (0) 25 minutes ago             practical_bartik
42a7627182bd        ubuntu             "/bin/bash"        3 hours ago        Exited (0) 3 hours ago               wonderful_jepsen
root@vuser-HVM-domU:~#
```

## Committing Changes in a Container to a Docker Image

When you start up a Docker image, you can create, modify, and delete files just like you can with a virtual machine. The changes that you make will only apply to that container. You can start and stop it, but once you destroy it with the **docker rm** command, the changes will be lost for good.

This section shows you how to save the state of a container as a new Docker image.

After installing Node.js inside the Ubuntu container, you now have a container running off an image, but the container is different from the image you used to create it. But you might want to reuse this Node.js container as the basis for new images later.

To do this, commit the changes to a new Docker image instance using the following command structure:

```
$ docker commit -m "What did you do to the image" -a "Author Name" container-id repository/new_image_name
```

The **-m** switch is for the commit message that helps you and others know what changes you made, while **-a** is used to specify the author. The **container ID** is the one you noted earlier in the tutorial when you started the interactive Docker session. Unless you created additional repositories on Docker Hub, the repository is usually your Docker Hub username.

For example, for the user **ybmrsr**, with the container ID of **d9b100f2f636**, the command would be:

```
$ docker commit -m "added node.js" -a "madhu" 6b6b363079ee ybmrsr/ubuntu-nodejs
```

**Note:** When you *commit* an image, the new image is saved locally, that is, on your computer. Later in this tutorial, you'll learn how to push an image to a Docker registry like Docker Hub so that it can be assessed and used by you and others.

After that operation is completed, listing the Docker images now on your computer should show the new image, as well as the old one that it was derived from:

```
root@vuser-HVM-domU:~# docker commit -m "added node.js" -a "madhu" 6b6b363079ee ybmrsr/ubuntu-nodejs
sha256:f49d31288afb089a1694da39f67f7d2d9cc40af6c22aac8491c25895db5de63
```

```
$ docker images
```

```
root@vuser-HVM-domU:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ybmsr/ubuntu-nodejs latest             f49d31288afb       About a minute ago 85.8MB
ubuntu              latest             ea4c82dcd15a       3 days ago         85.8MB
hello-world         latest             4ab4c602aa5e       6 weeks ago        1.84kB
root@vuser-HVM-domU:~#
```

In the above example, **ubuntu-nodejs** is the new image, which was derived from the existing ubuntu image from Docker Hub. The size difference reflects the changes that were made. In this example, the change was that Node.js was installed. Next time you need to run a container using Ubuntu with Node.js pre-installed, you can just use the new image.

You can also build images from a **Dockerfile**, which lets you automate the installation of software in a new image.

Now let's share the new image with others so they can create containers from it.

## Pushing Docker Images to a Docker Repository

The next logical step after creating a new image from an existing image is to share it with a select few of your friends, the whole world on Docker Hub, or another Docker registry that you have access to. To push an image to Docker Hub or any other Docker registry, you must have an account there.

This section shows you how to push a Docker image to Docker Hub. how to create your own private Docker registry will learn later. As of now we will use public docker hub repository to push and pull docker your docker images.

To push your image, first log into Docker Hub:

```
$ docker login -u docker-registry-username
```

```
root@vuser-HVM-domU:~# docker login -u ybmsr
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

You'll be prompted to authenticate using your Docker Hub password. If you specified the correct password, authentication should succeed.

**Note:** If your Docker registry username is different from the local username you used to create the image, you will have to tag your image with your registry username. For the example given in the last step, you would type:

```
$ docker tag ybmsr/ubuntu-nodejs docker-registry-username/ubuntu-nodejs
```

Then you can push your own image using:

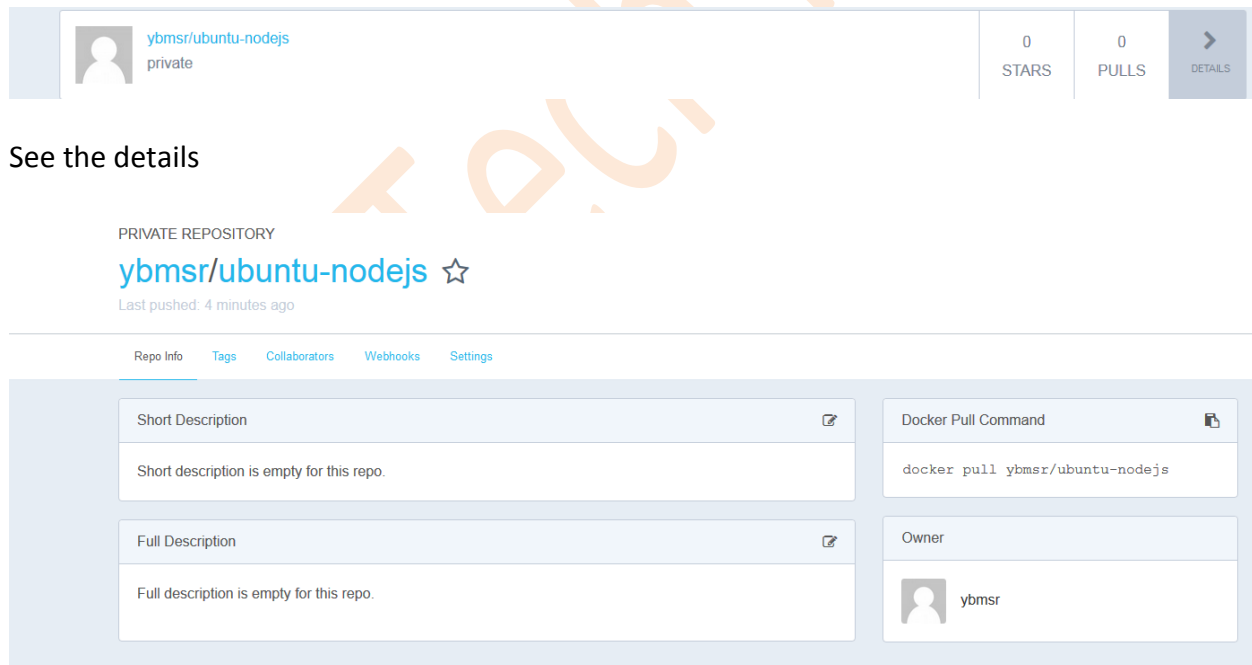
```
$ docker push docker-registry-username/ubuntu-nodejs
```

In my nodejs images I am pushing to docker hub.

```
$ docker push ybmsr/ubuntu-nodejs
```

```
root@vuser-HVM-domU:~# docker push ybmsr/ubuntu-nodejs
The push refers to repository [docker.io/ybmsr/ubuntu-nodejs]
76c033092e10: Mounted from library/ubuntu
2146d867acf3: Mounted from library/ubuntu
ae1f631f14b7: Mounted from library/ubuntu
102645f1cf72: Mounted from library/ubuntu
latest: digest: sha256:2cf4d5301cc4167111c83ede8f1d896045434b5cbd945c17b399a97eb7a48097 size: 1150
```

Now let's verify the my image is pushed to docker hub or not. Open the browser and login to docker hub with your credentials.



The screenshot shows the Docker Hub interface for a private repository named 'ybmsr/ubuntu-nodejs'. At the top, it indicates '0 STARS' and '0 PULLS'. Below this, the repository is identified as 'PRIVATE REPOSITORY' and 'ybmsr/ubuntu-nodejs' with a star icon. It notes 'Last pushed: 4 minutes ago'. The 'Repo Info' tab is selected, showing fields for 'Short Description' and 'Full Description', both of which are empty. To the right, the 'Owner' is listed as 'ybmsr' with a profile icon. The 'Docker Pull Command' is displayed as 'docker pull ybmsr/ubuntu-nodejs'.

And check the tags tab it will show the tag name, compared size of the image and Last updated date.



PRIVATE REPOSITORY

ybmsr/ubuntu-nodejs ☆

Last pushed: 5 minutes ago

---

[Repo Info](#) [Tags](#) [Collaborators](#) [Webhooks](#) [Settings](#)

Tag Name	Compressed Size	Last Updated
latest	0 B	5 minutes ago

**Note:** if you try without login it will through the error is **unauthorized : authentication required**.

JMS Tech Home