**Docker commands:**

1. Want to check docker version use below command.

## $ docker –v (or) $ docker --version

1. Check complete docker information use below commands.

## $ docker info

1. Check what are all the option can we use in docker use below command.

## $ docker

1. List out the docker images use below command.

## $ docker images

1. If you want to search any docker images use below command.

## $ docker search [option] Ex: docker search Ubuntu

1. 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

1. 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**

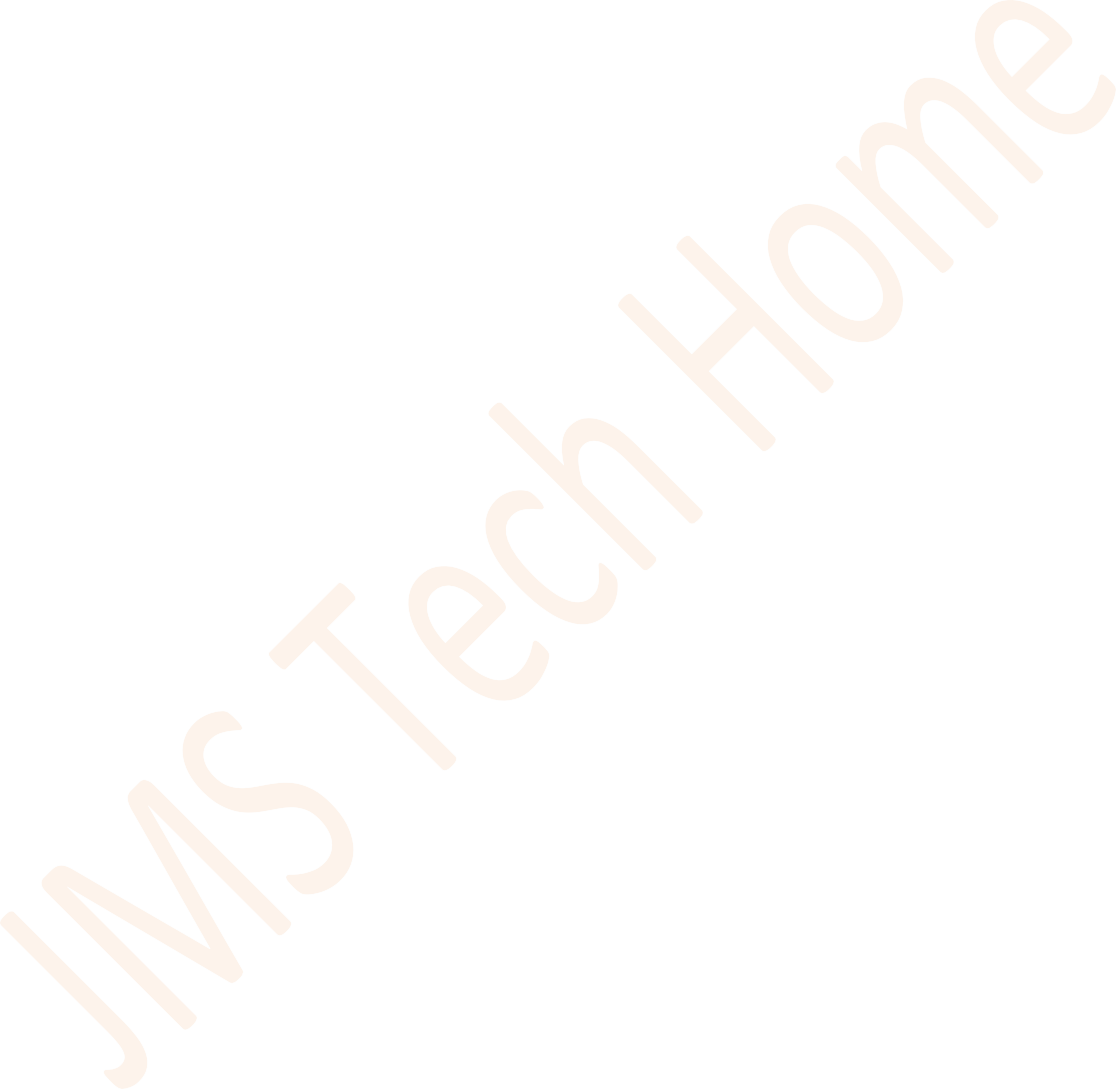
Note: if we want check outside access port on the running container port use below command.

**$ 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.

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

**$ docker ps**

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

**$ docker ps –a**

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

**$ docker rm <containername>**

1. remove the docker images from the list.

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

1. check oly docker container ID’s use below command.

**$ docker ps –q**

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

**$ docker ps –l**

1. to kill running docker container use below command.

**$ docker kill <container id>**

1. 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.

1. To run the stopped container use below command.

**$ docker ps -a**

**$ docker start <container id>**

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

**$ docker ps -a**

**$ docker logs <container id>**

**$ docker logs 35nfdfw453** (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 - -follow - -tail 1 <container ID>**

1. 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.

1. Build a custom docker image use below command.

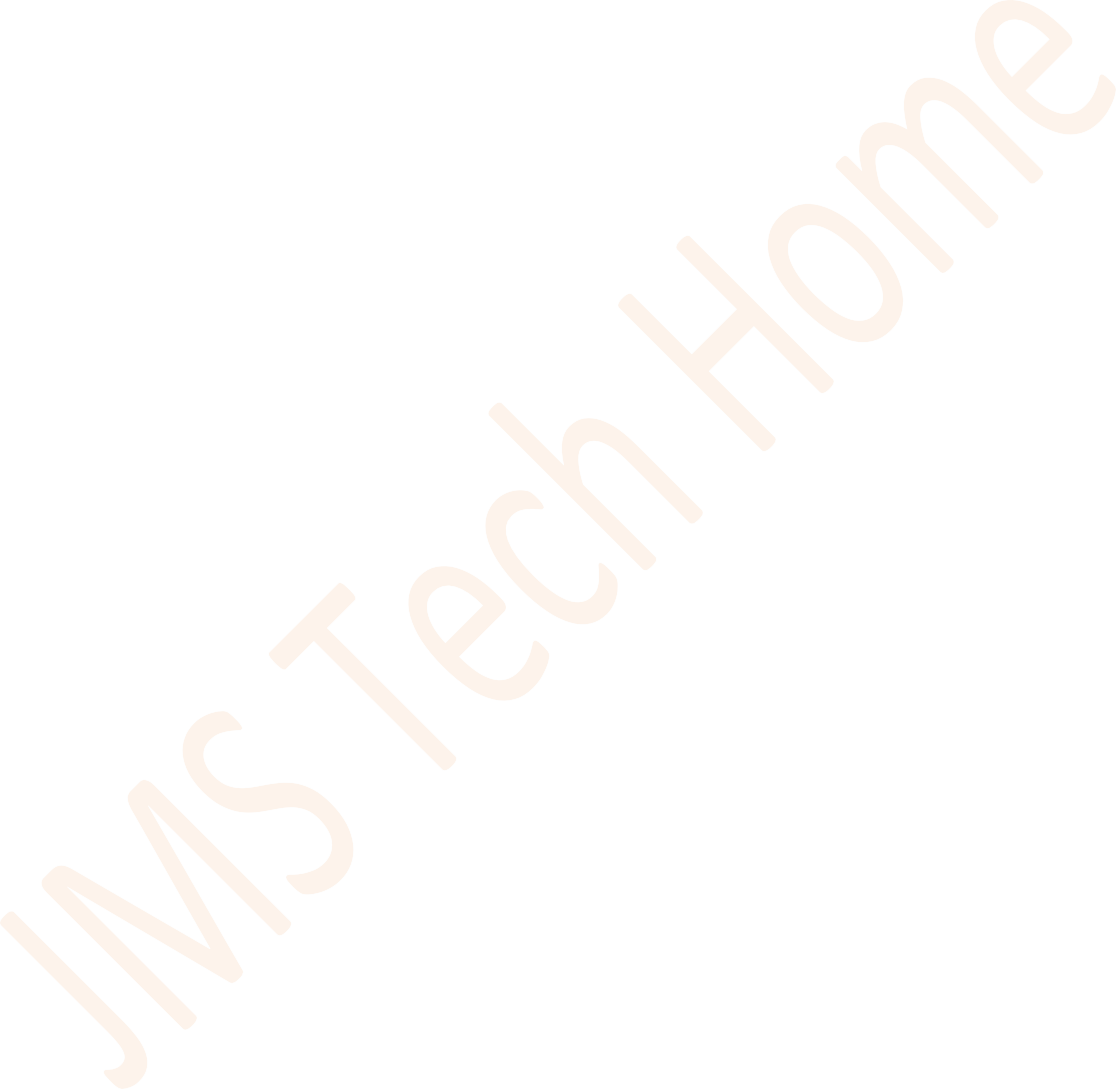
**$docker build –it ybmsr/hello:v1 .**

Without cache

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

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

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

1. 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"

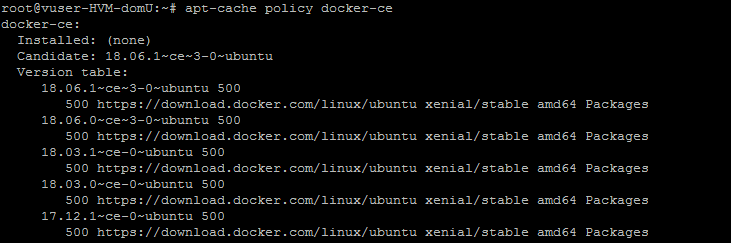
Next, update the package database with the Docker packages from the newly added repo:

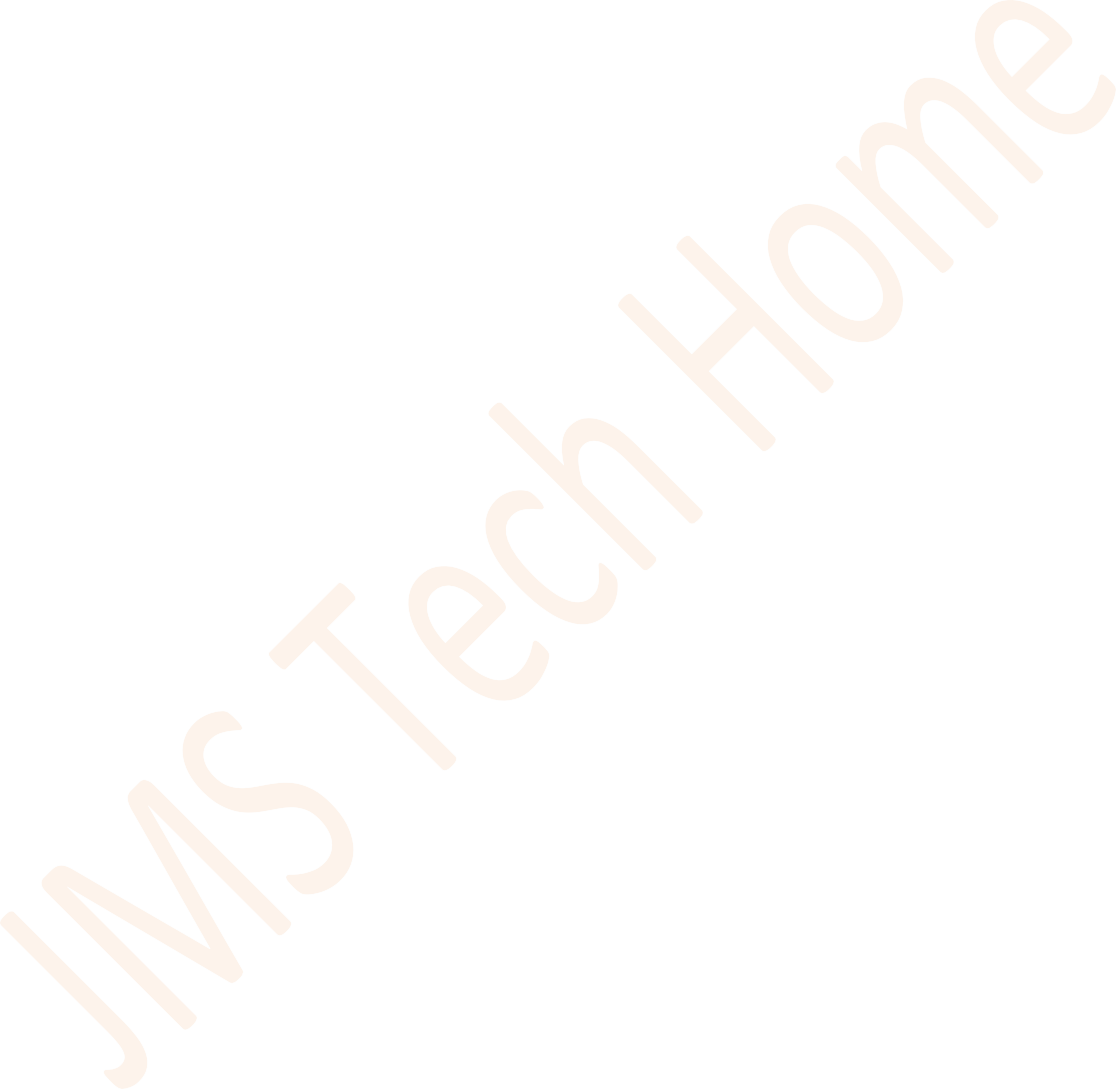
$ sudo apt-get update

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:



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

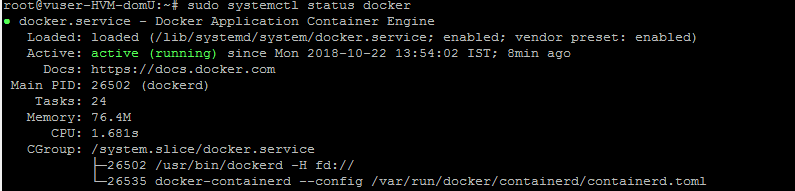
Finally, install Docker:

$ 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

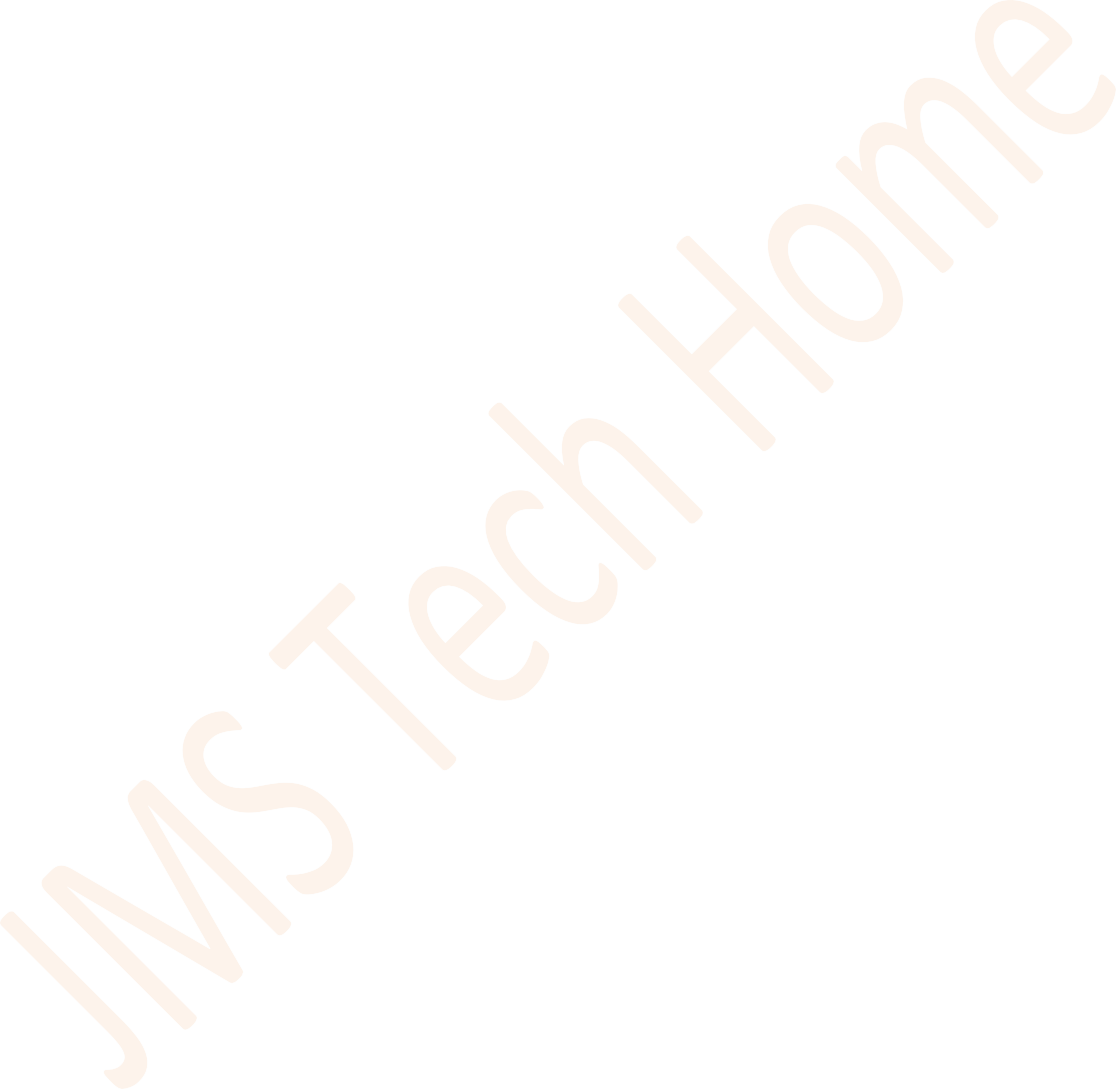
The output should be similar to the following, showing that the service is active and running:



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>

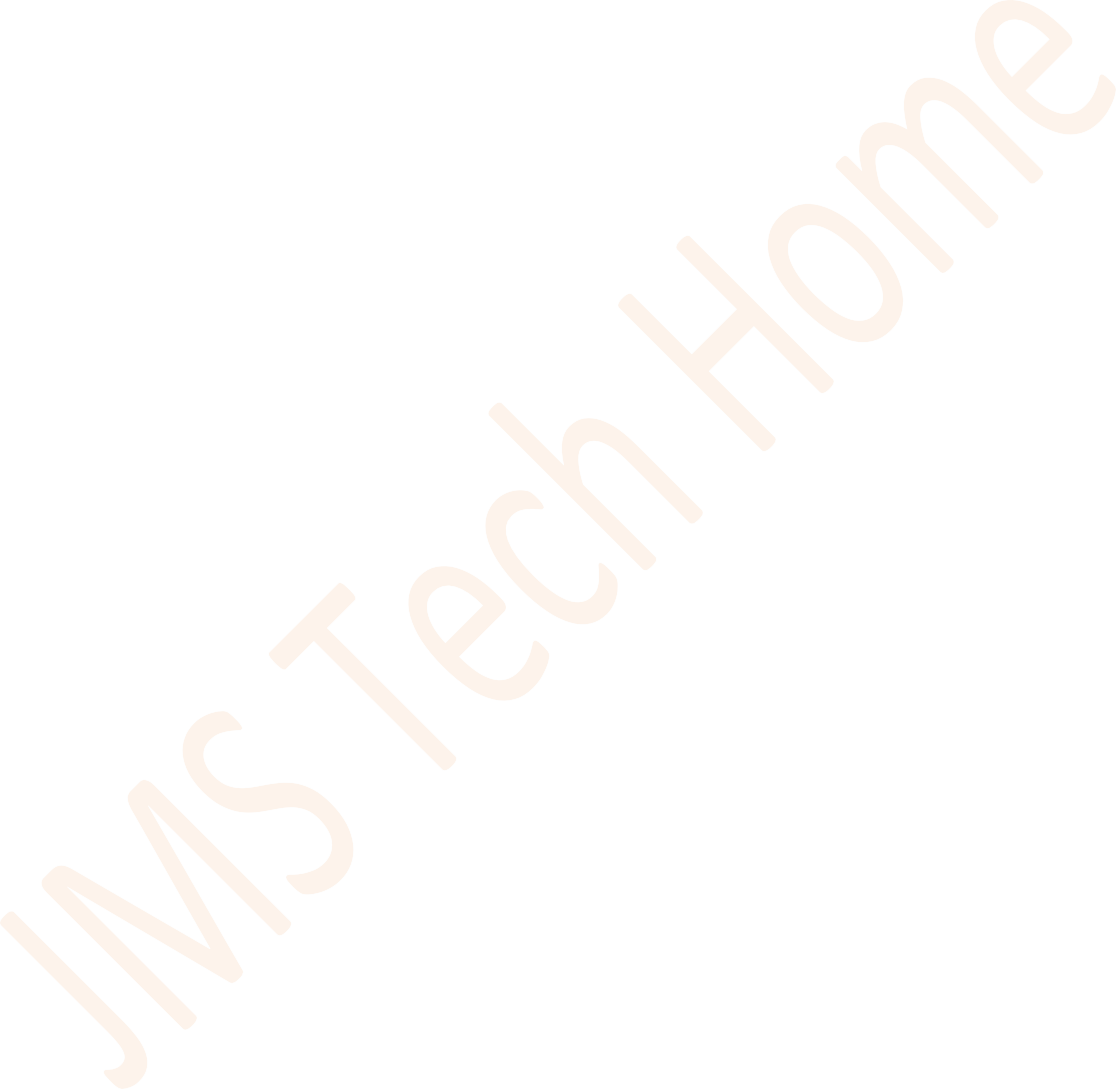
# Using the Docker Command

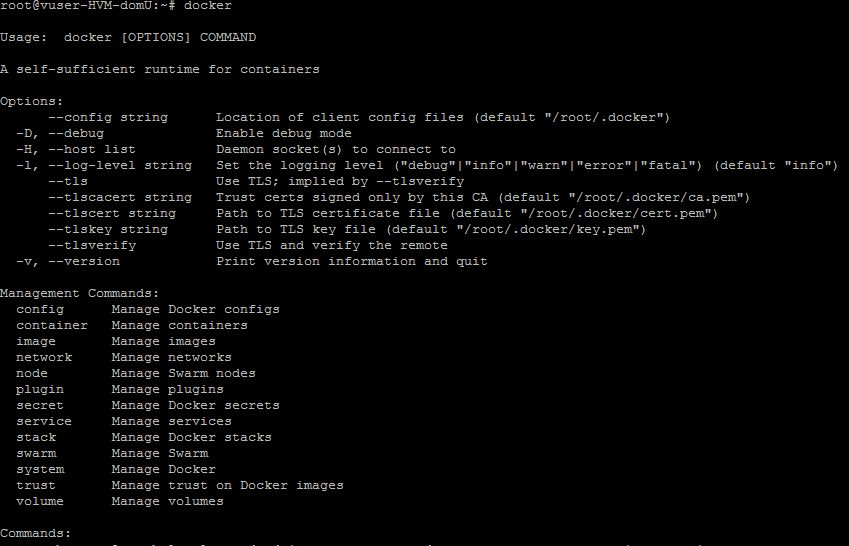
With *Docker* installed and working, now's the time to become familiar with the command line utility. Using *docker* consists of passing it a chain of options and commands followed by arguments. The syntax takes this form:

$ docker [option] [command] [arguments]

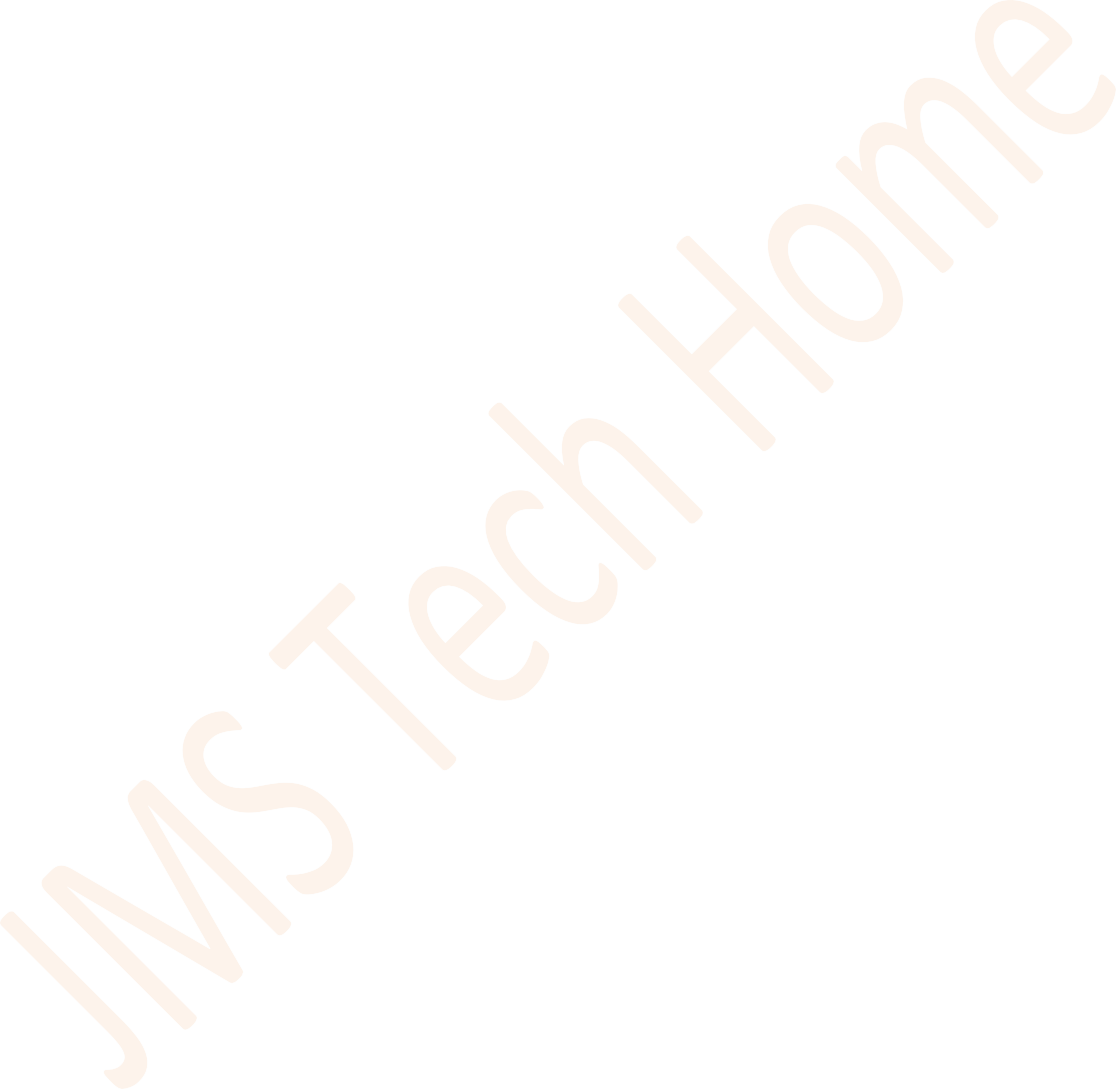
To view all available subcommands, type:

$ docker

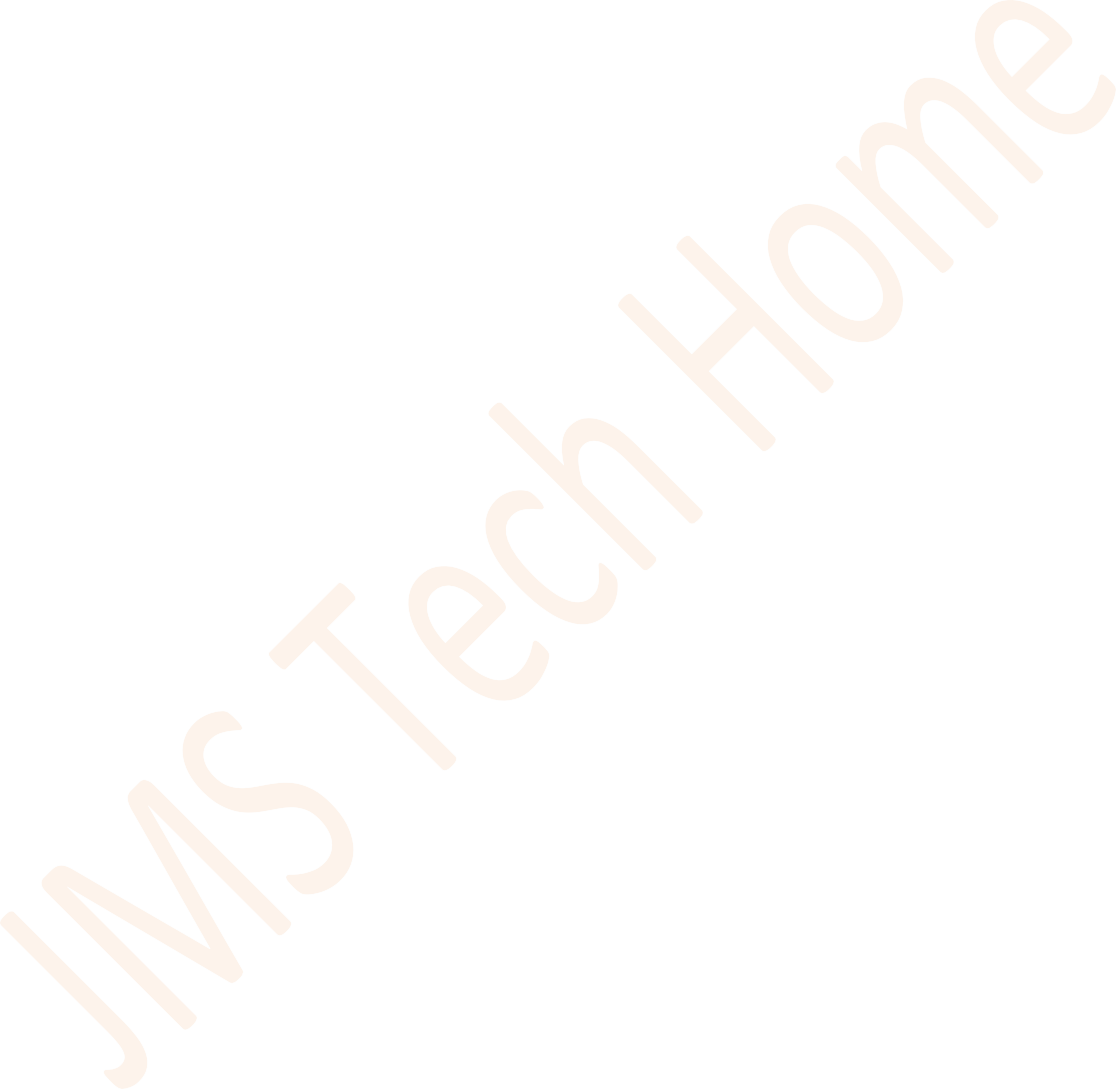
As of Docker 18.06.1, the complete list of available subcommands includes:

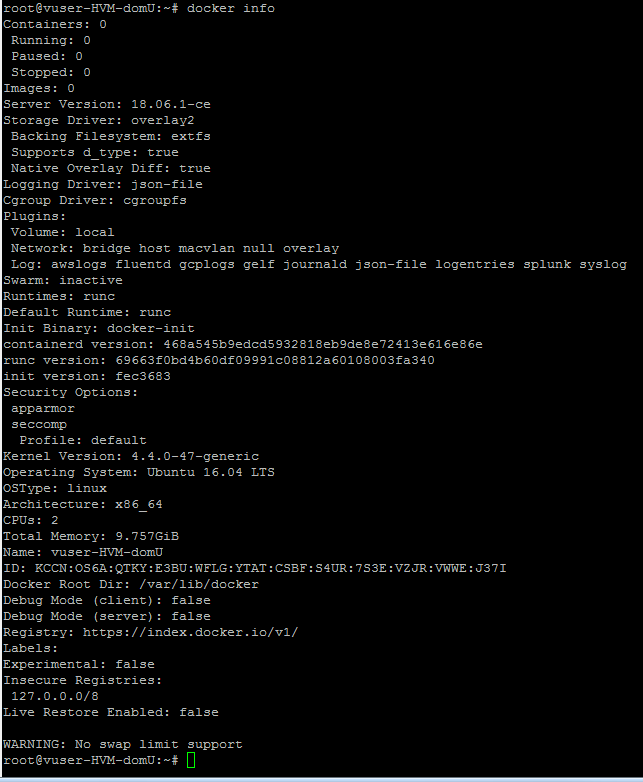




To view system-wide information about Docker, use:

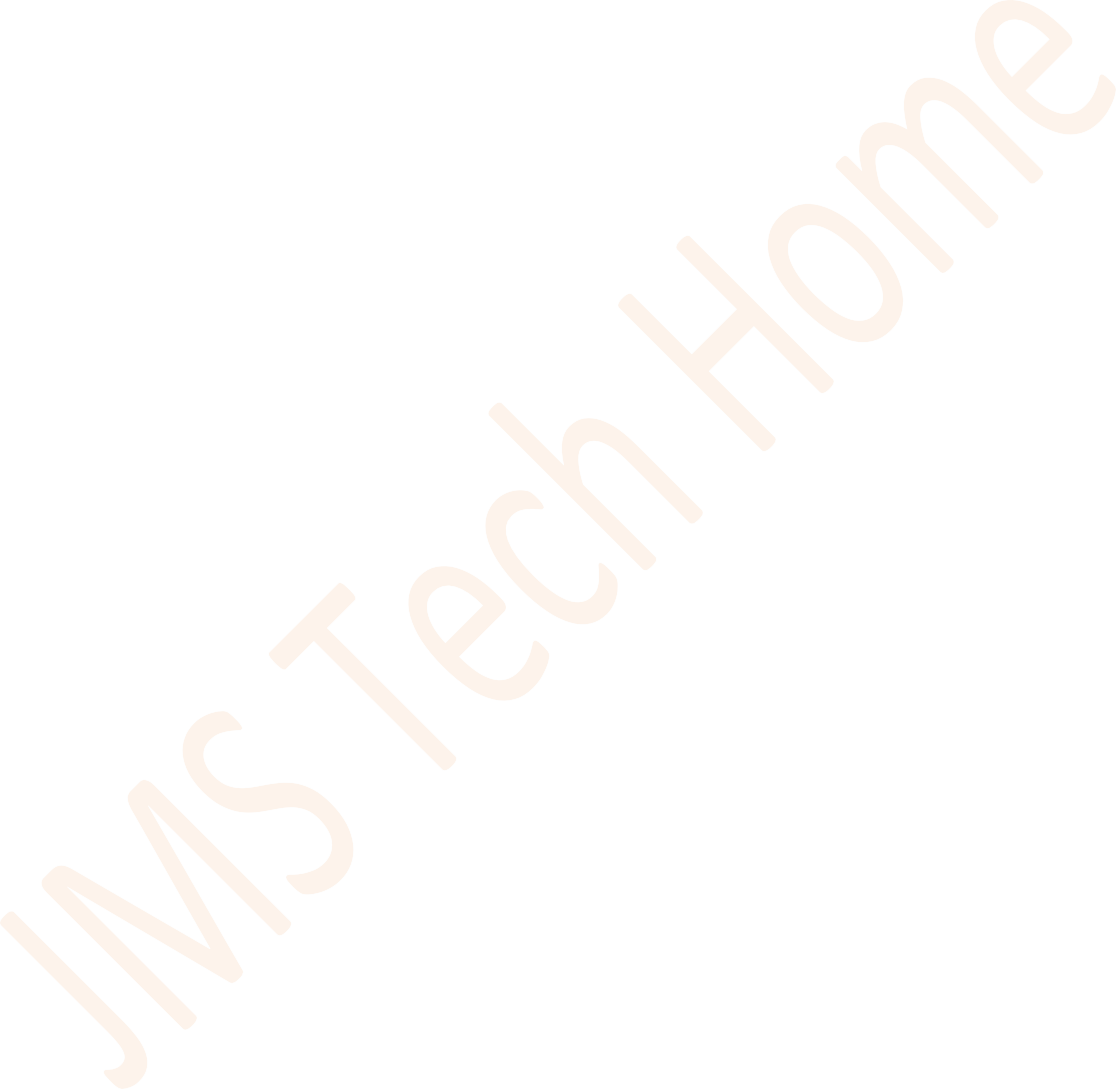
$ docker info





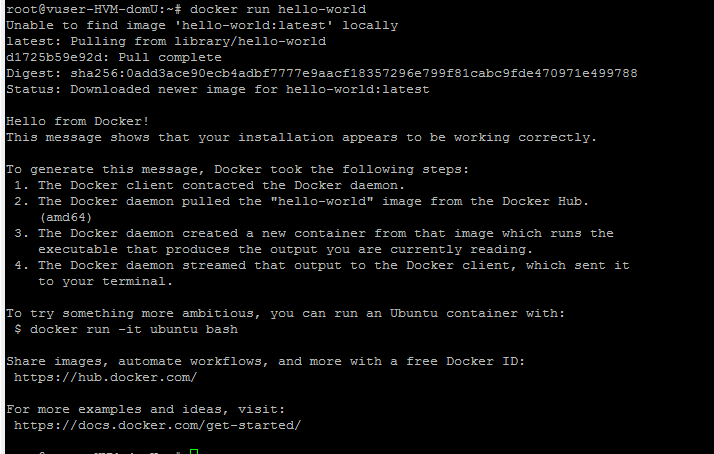
# Working with Docker Images

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:

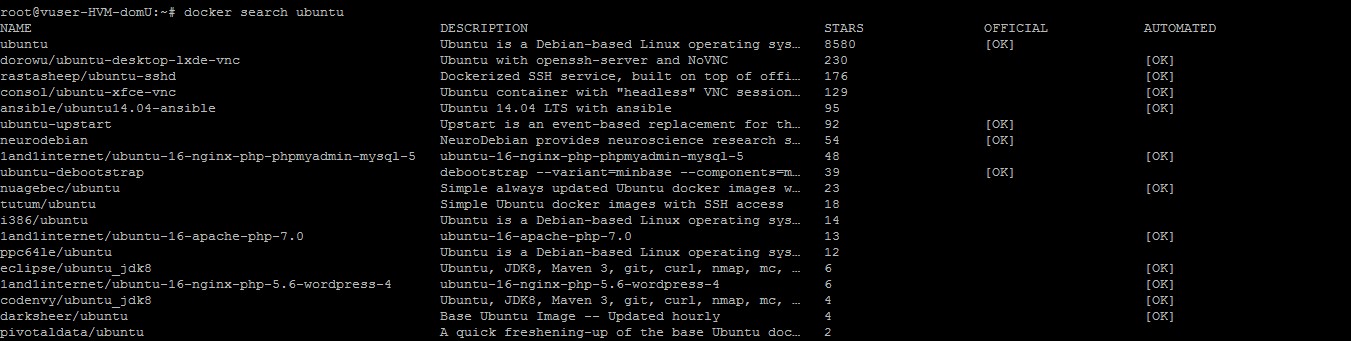


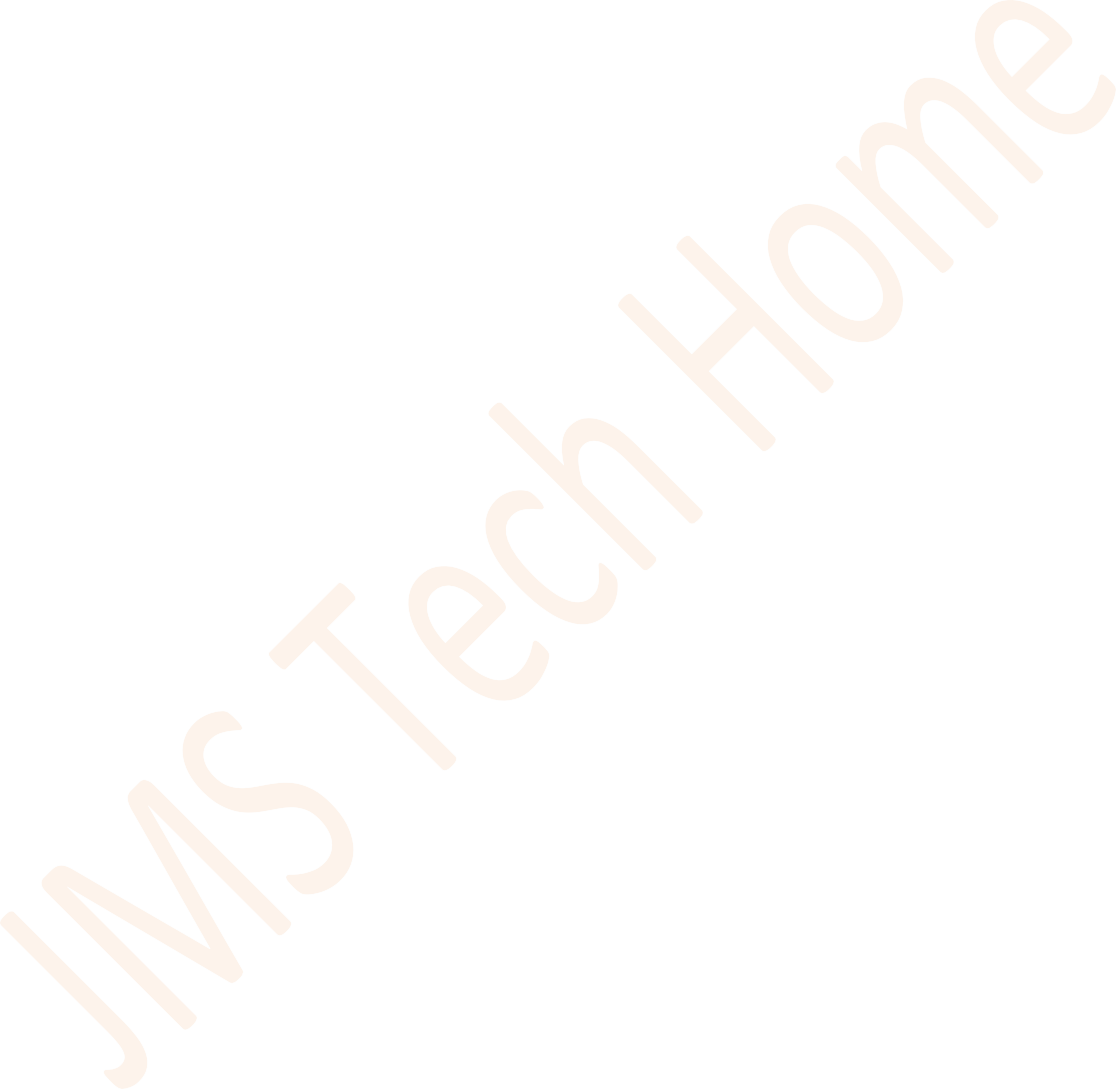
You can search for images available on Docker Hub by using the *docker* command with the

*search* subcommand. For example, to search for the Ubuntu image, type:

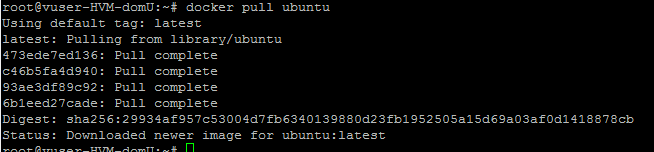
$ docker search ubuntu

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:



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



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

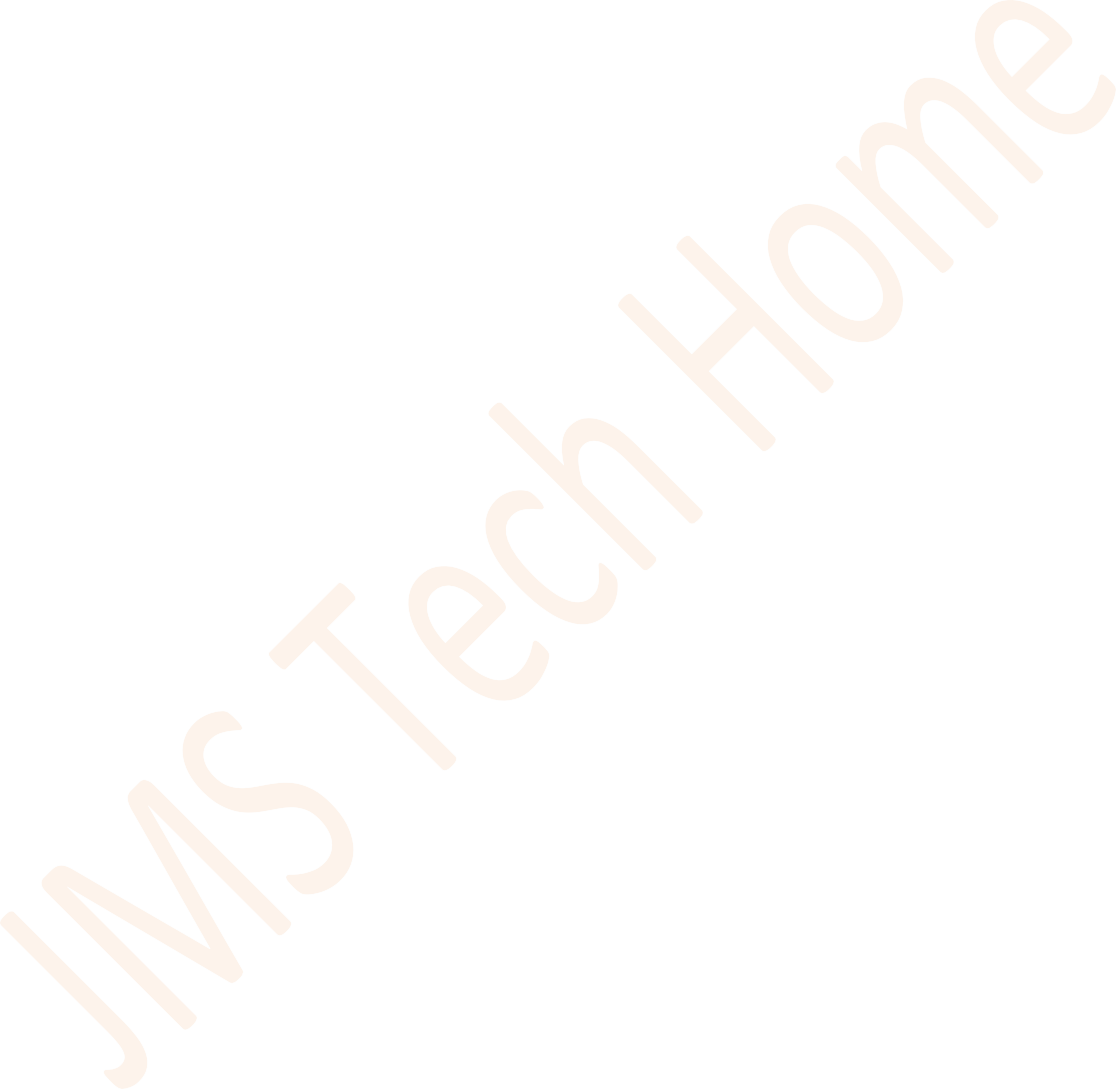
The output should look similar to the following:



# 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 the command, you will open up the shell interface of a second ubuntu container.



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

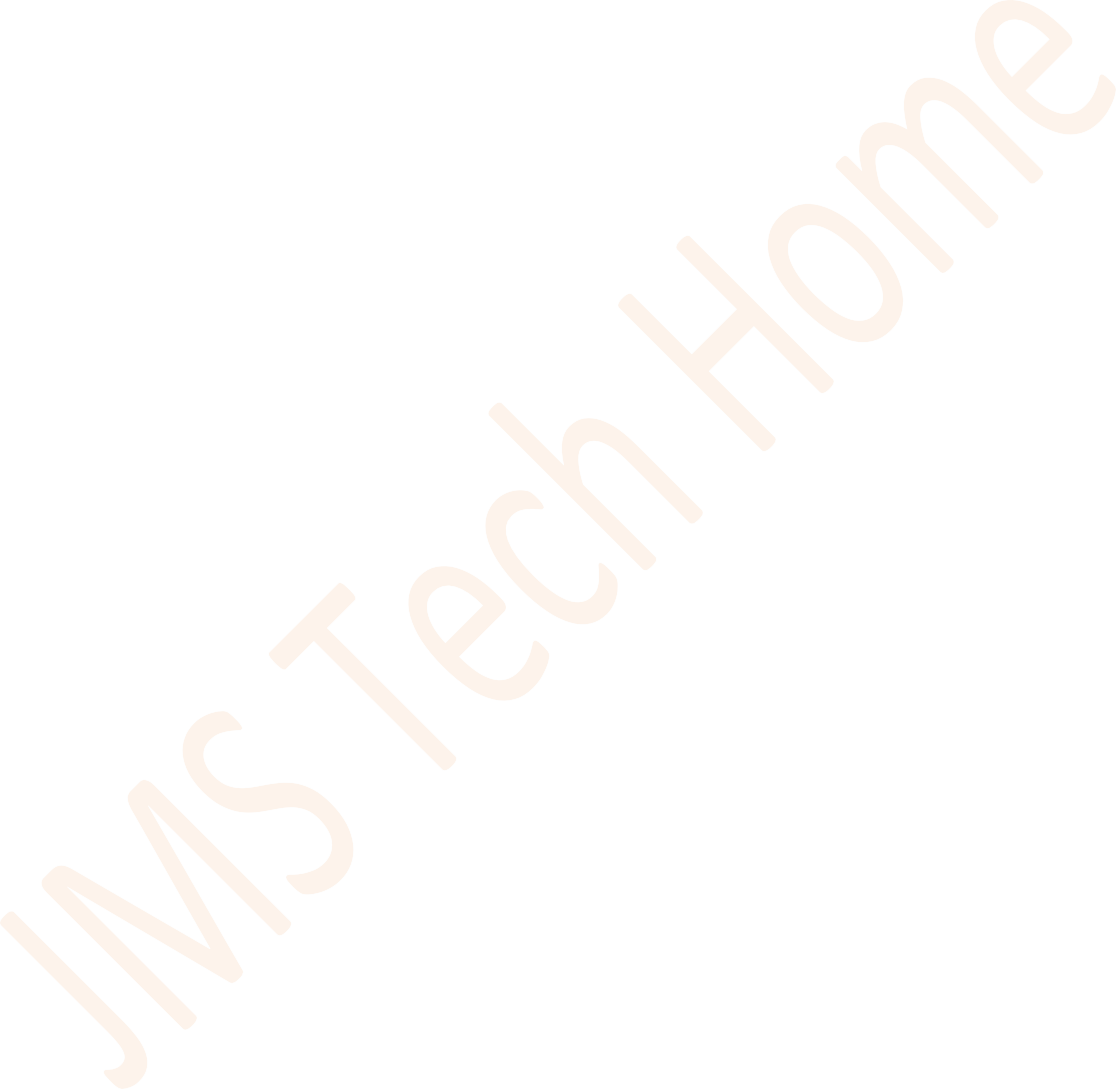
Then install any application in it. Let's install Node.js:

$ 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:

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

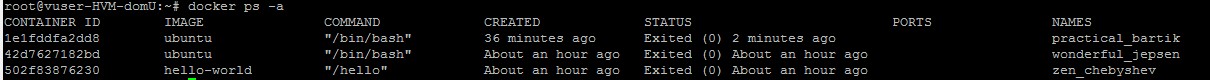
You will see output similar to the following:



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



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

$ docker ps -l

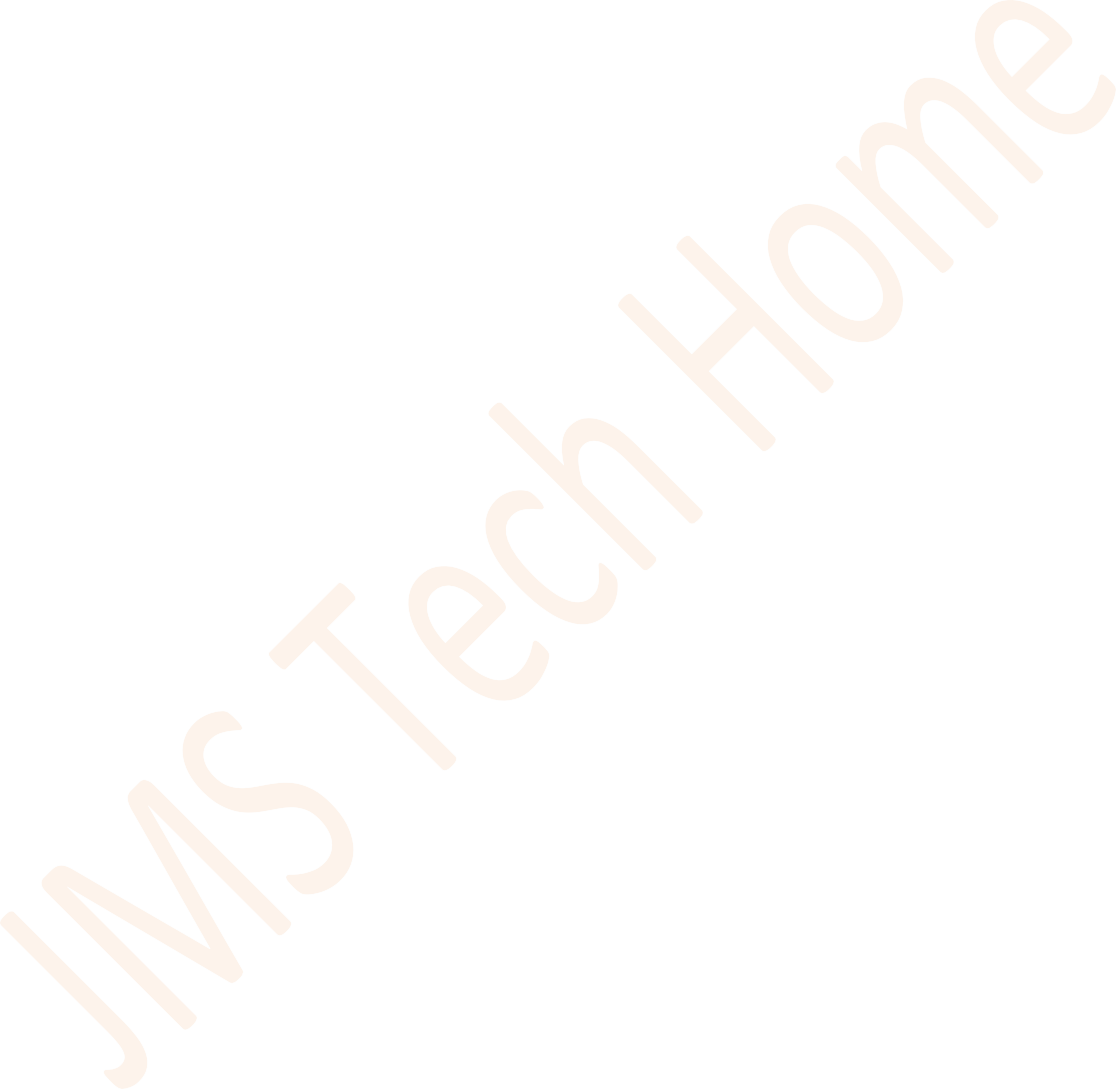


To start a stopped container, use docker start, followed by the **container ID** or the container's name. Let's start the Ubuntu-based container with the ID of **9b0db8a30ad1**:

$ docker start 1e1fddfa2dd8



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

$ docker ps



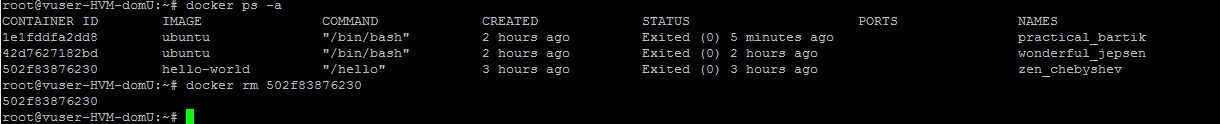
To stop a running container, use docker stop, followed by the **container ID** or **name**. This time, we'll use the name that Docker assigned the container, which is *practical\_bartik*:

$ docker stop *practical\_bartik*



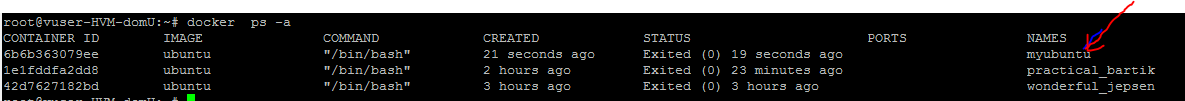
Once you've decided you no longer need a container anymore, remove it with the ***docker rm*** command, again using either the **container ID** or the **name**. Use the **docker ps -a** command to find the container ID or name for the container associated with the **hello-world** image and remove it.

$ docker rm 502f83876230

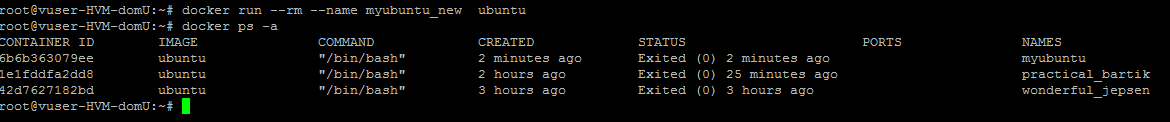


You can start a new container and give it a name using the --name switch. You can also use the

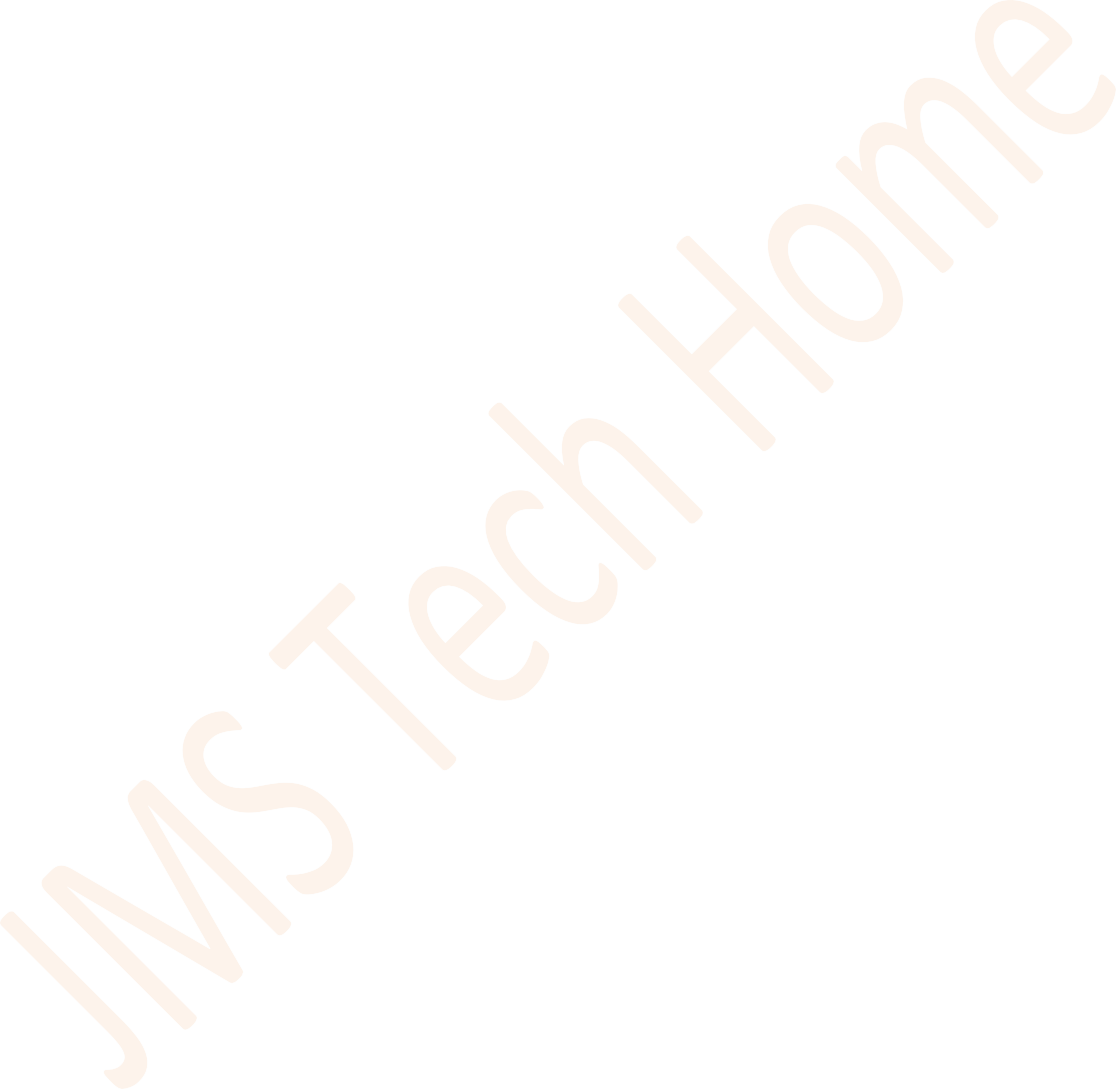
--rm switch to create a container that removes itself when it's stopped.



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



# 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 **ybmsr**, with the container ID of d9b100f2f636, the command would be:

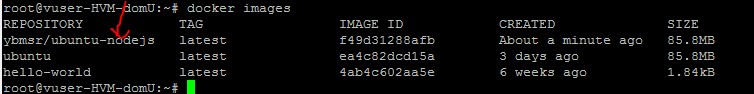
$ docker commit -m "added node.js" -a "madhu" 6b6b363079ee ybmsr/ubuntu-nodejs

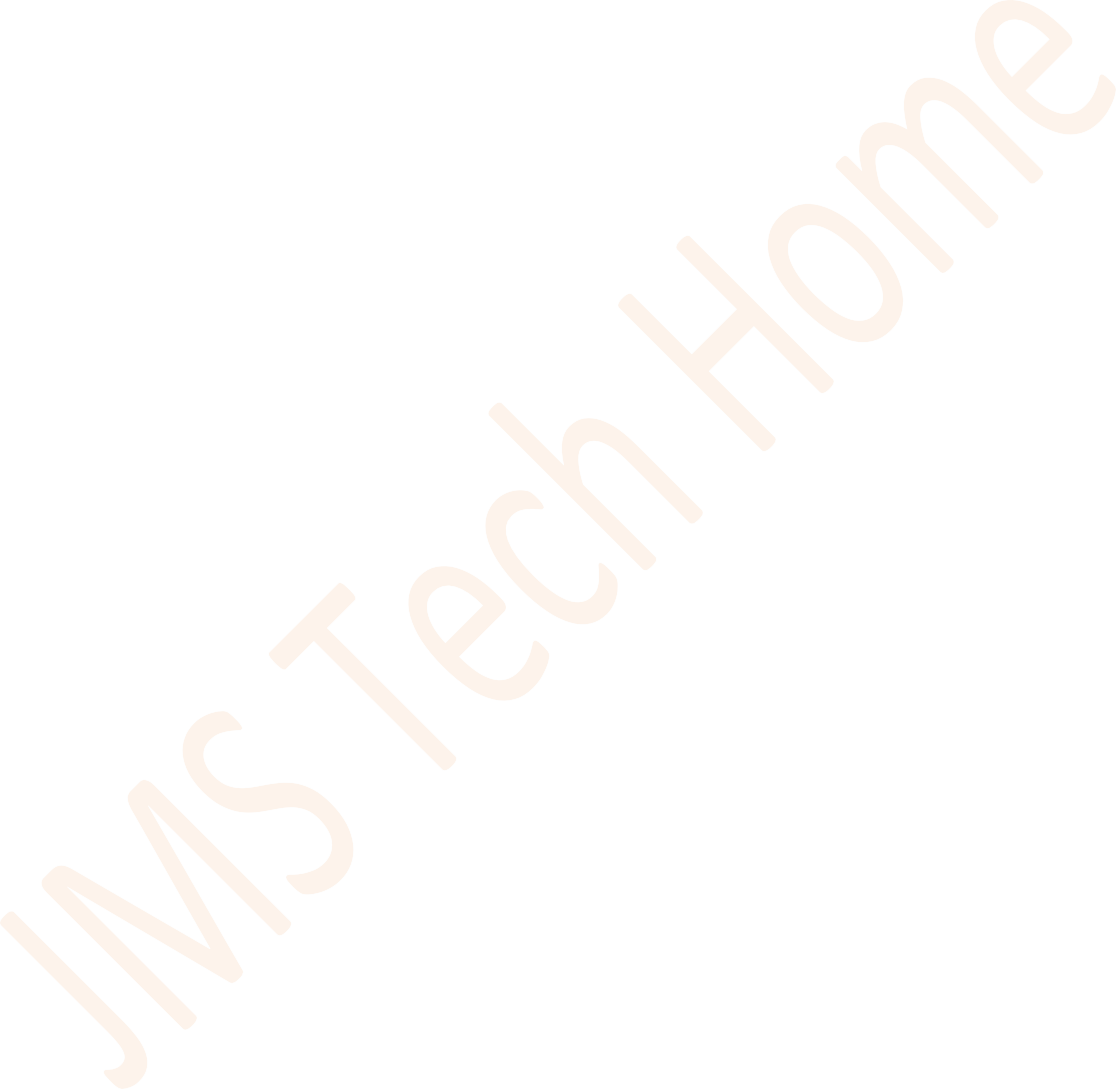
**Note:** When you *commit* an image, the new image is saved locally, that is, on your computer. Later in this tutorial, you'll learn how to push an image to a Docker registry like Docker Hub so that it can be assessed and used by you and others.

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:



$ docker images



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.

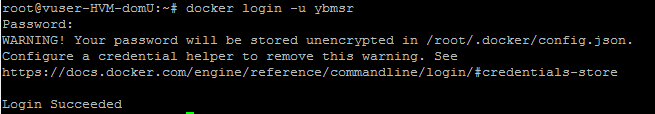
# Pushing Docker Images to a Docker Repository

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 know 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



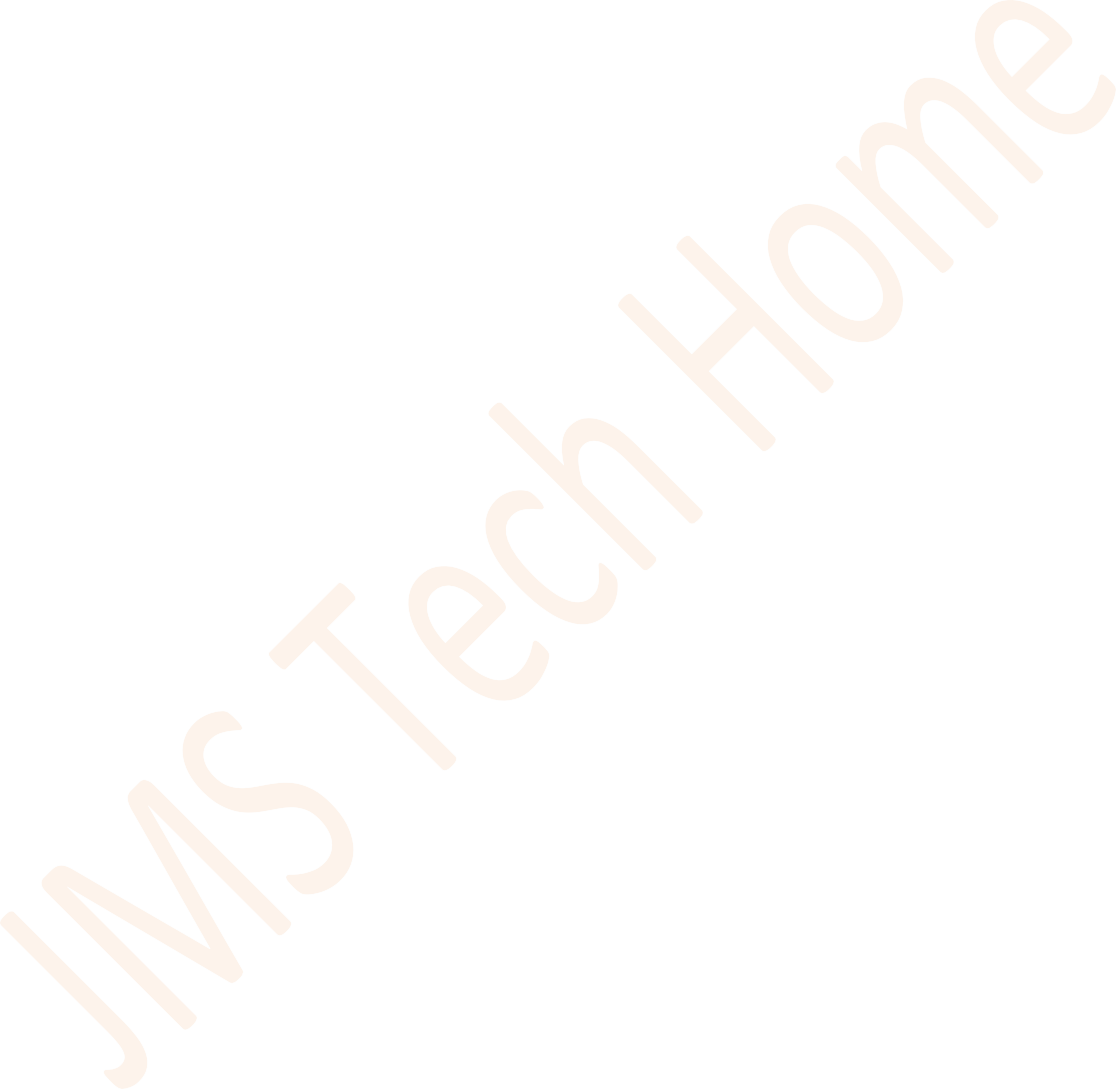
You'll be prompted to authenticate using your Docker Hub password. If you specified the correct password, authentication should succeed.

**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:

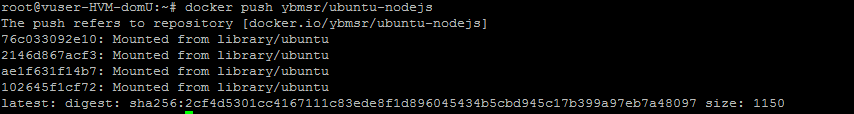
$ docker tag ybmsr/ubuntu-nodejs docker-registry-username/ubuntu-nodejs

Then you can push your own image using:

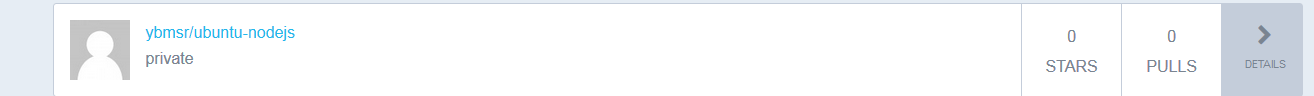
$ docker push docker-registry-username/ubuntu-nodejs

In my nodejs images I am pushing to docker hub.

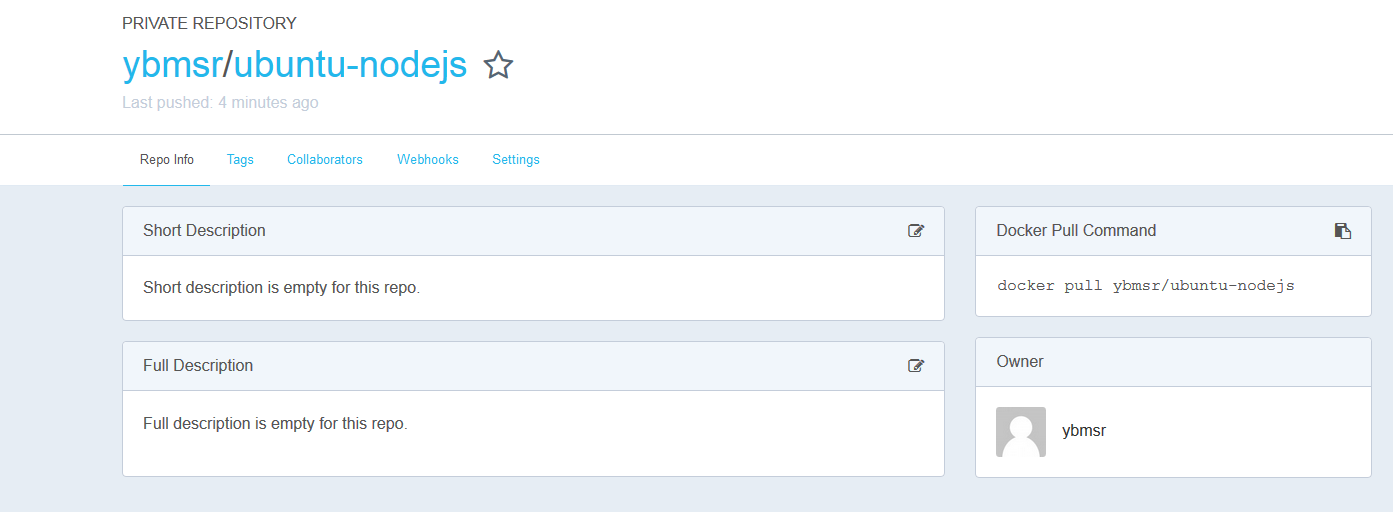
$ docker push ybmsr/ubuntu-nodejs



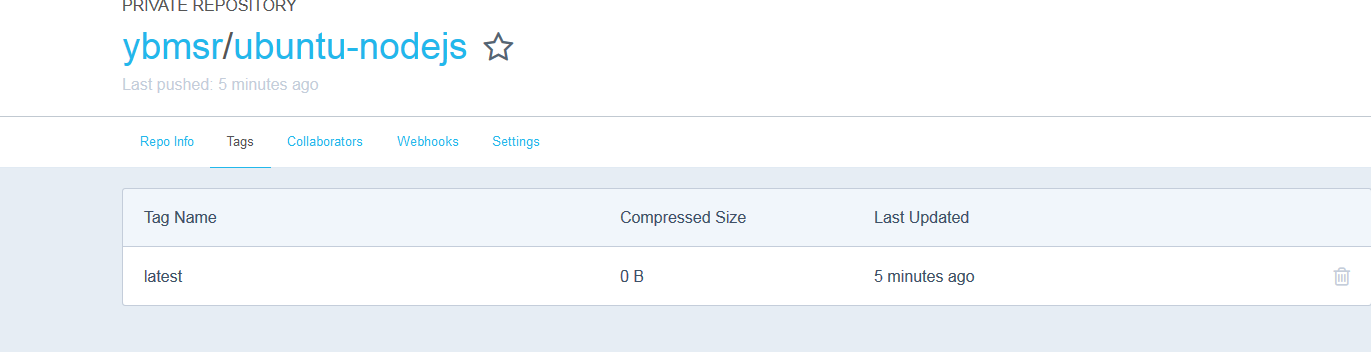
Now let’s verify the my image is pushed to docker hub or not. Open the browser and login to docker hub with your credentials.

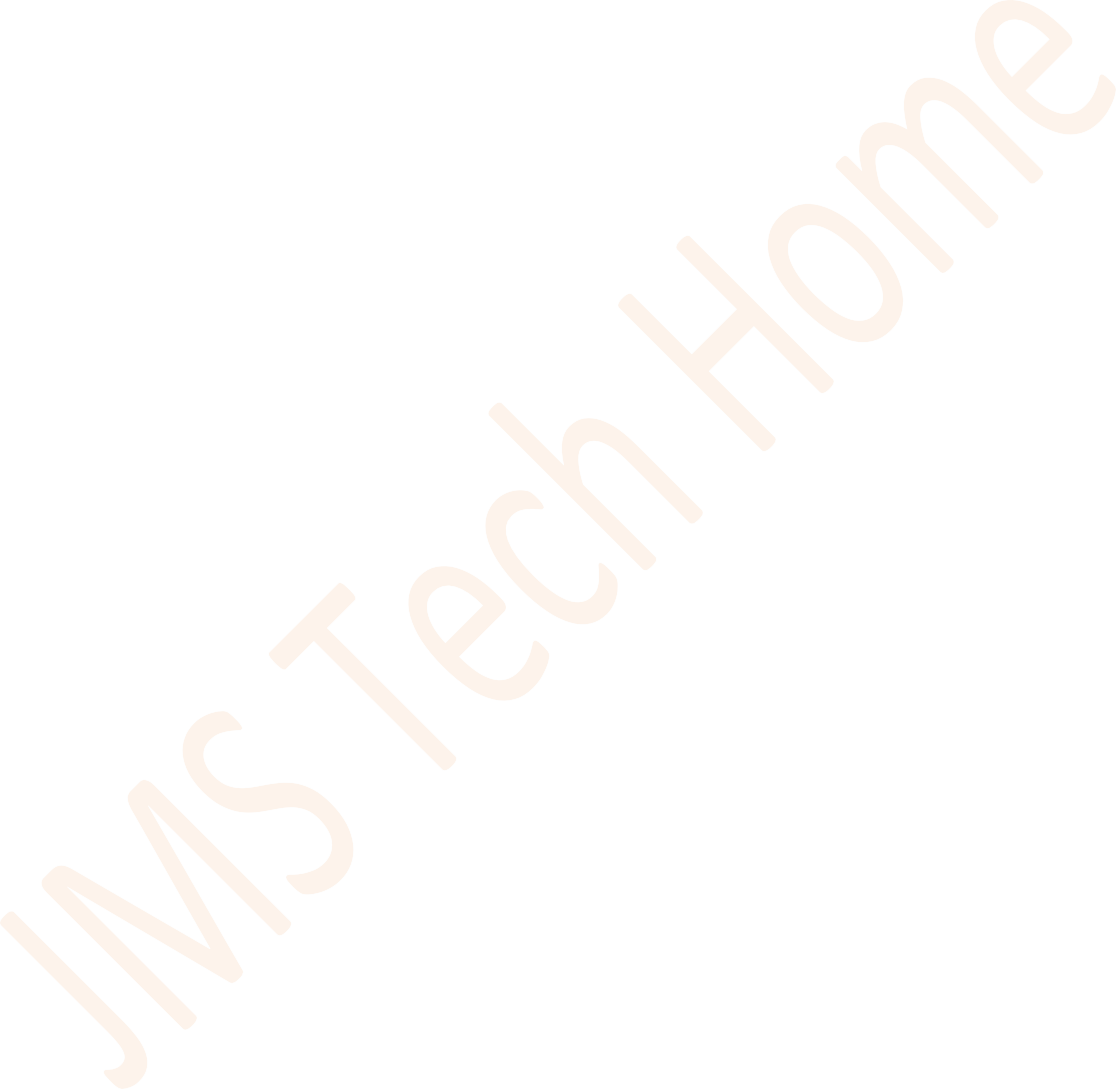


See the details



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



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

