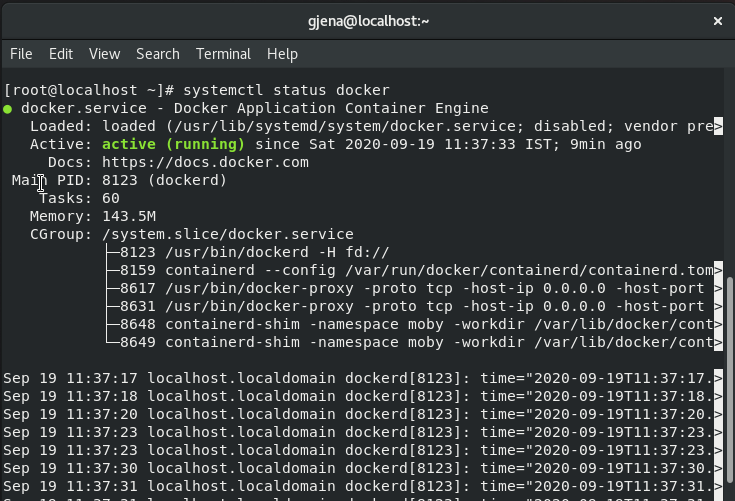
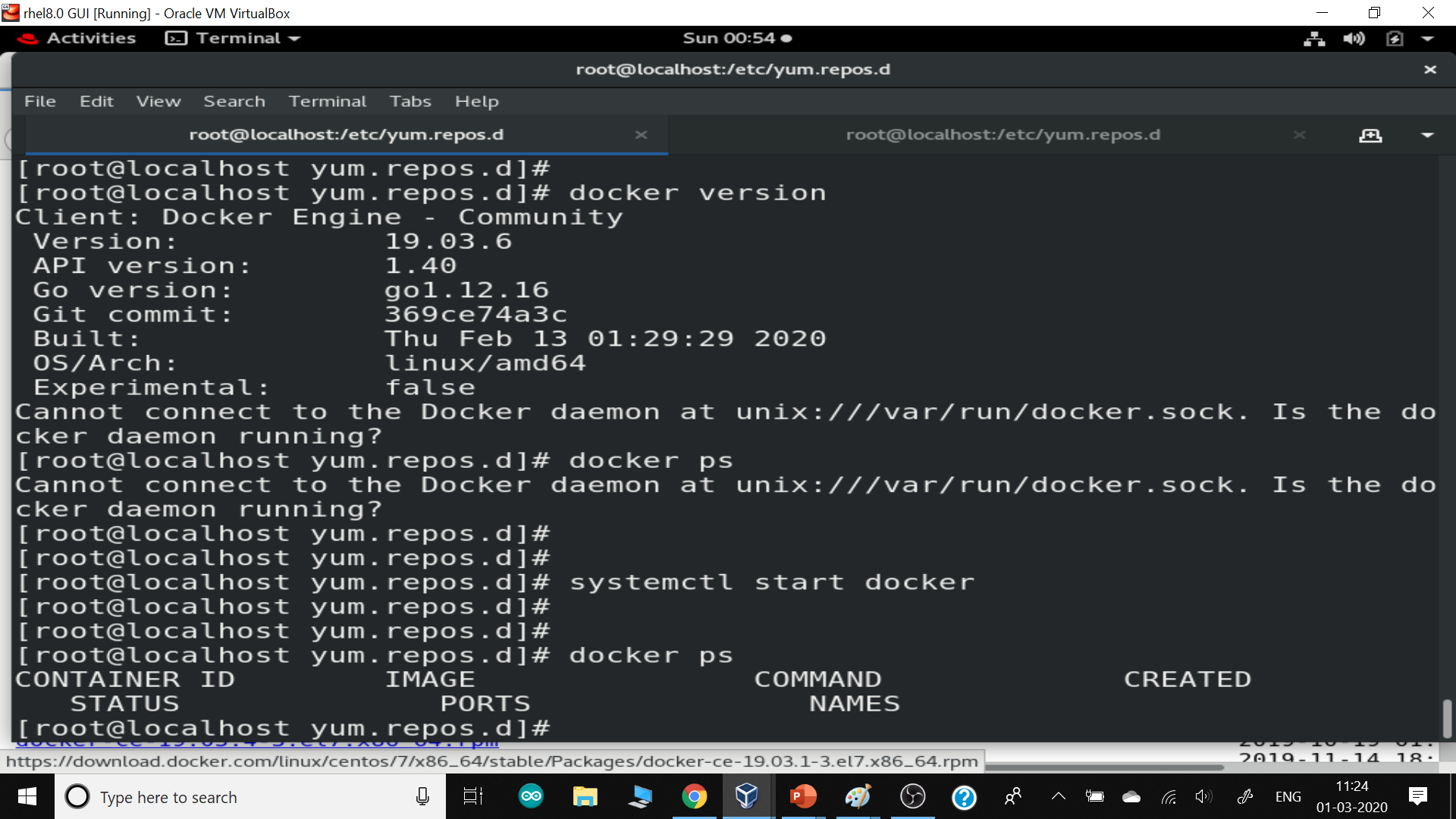
## Docker Tutorial

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With the help of Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker’s methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production environment.

**Commands:**





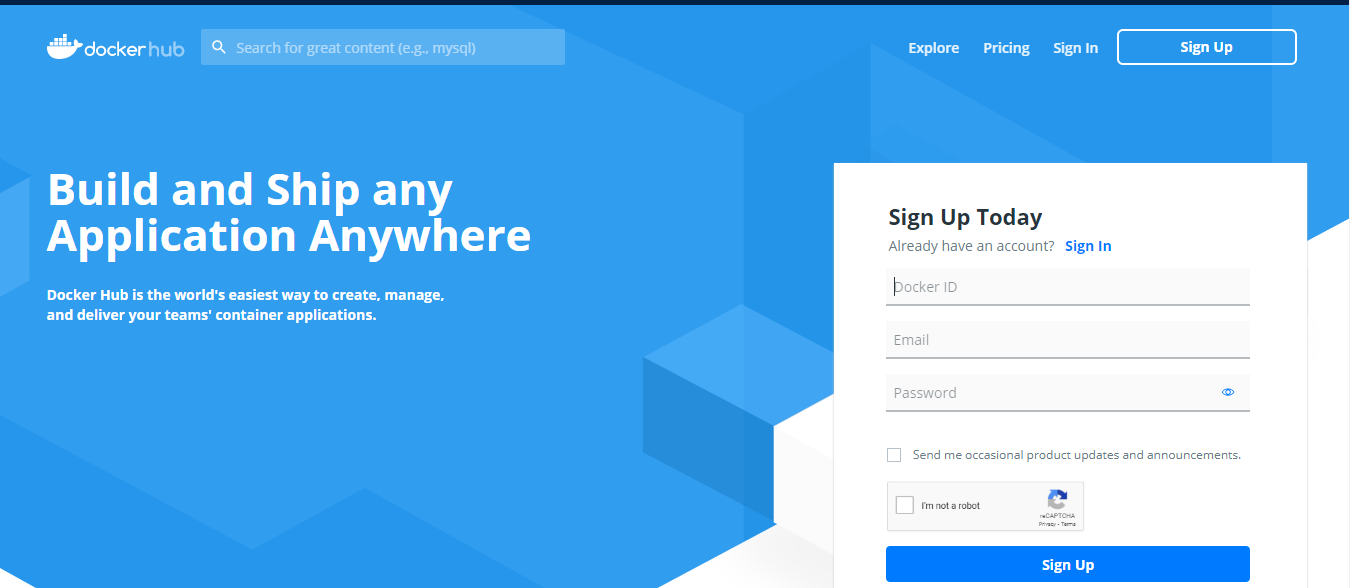
## Docker - Hub

Docker Hub is a registry service on the cloud that allows you to download Docker images that are built by other communities. You can also upload your own Docker built images to Docker hub. In this chapter, we will see how to download and the use the Jenkins/Maven Docker image from Docker hub.

The official site for Docker hub is

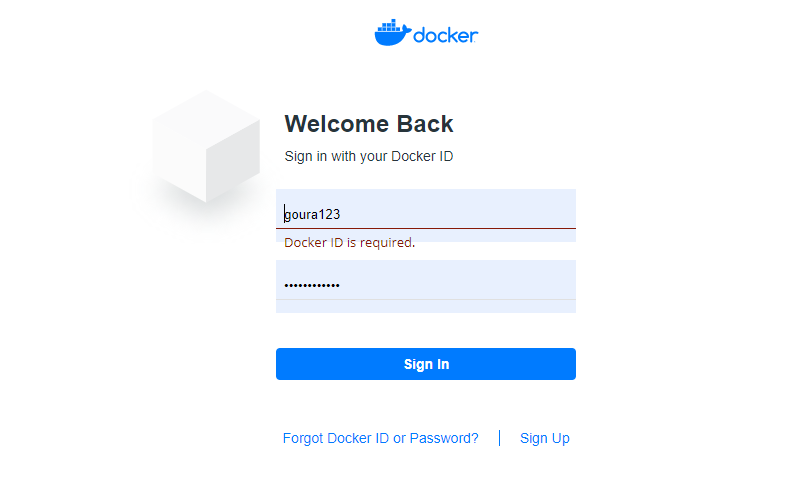
<https://www.docker.com/community-edition#/add_ons>

**Step 1** − First you need to do a simple sign-up on Docker hub.



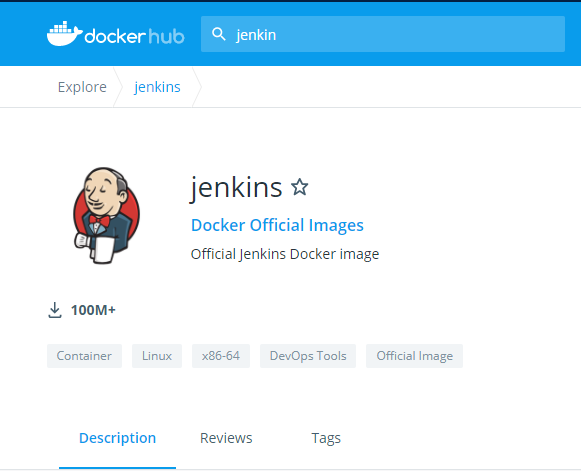
*Fig-1*

**Step 2** − Once you have signed up, you will be logged into Docker Hub.



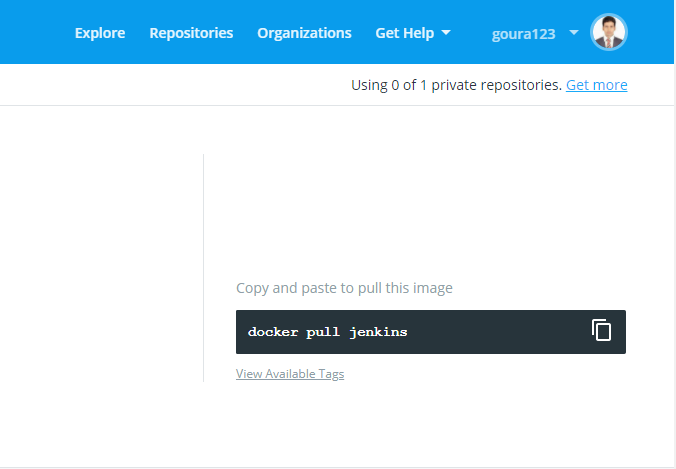
*Fig-2*

**Step 3** − Next, let’s browse and find the Jenkins image.



*Fig-3*

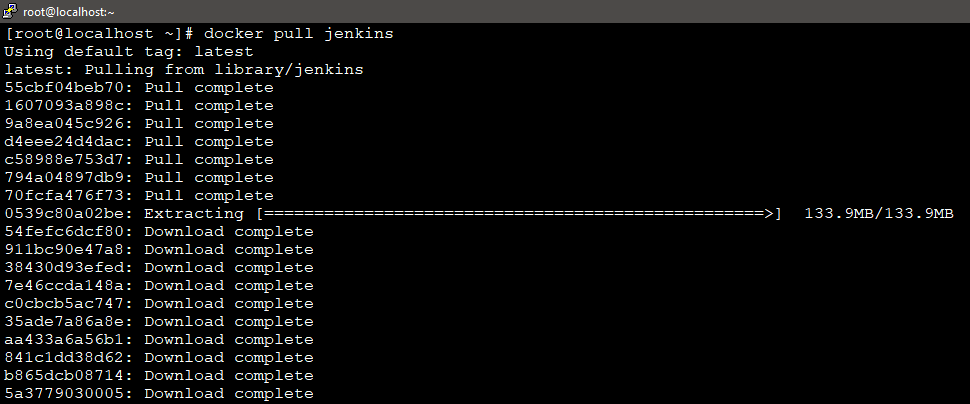
**Step 4** − you can see the Docker pull command in the same page in the right side. This will be used to download the Maven image onto the local Linux server.



*Fig-4*

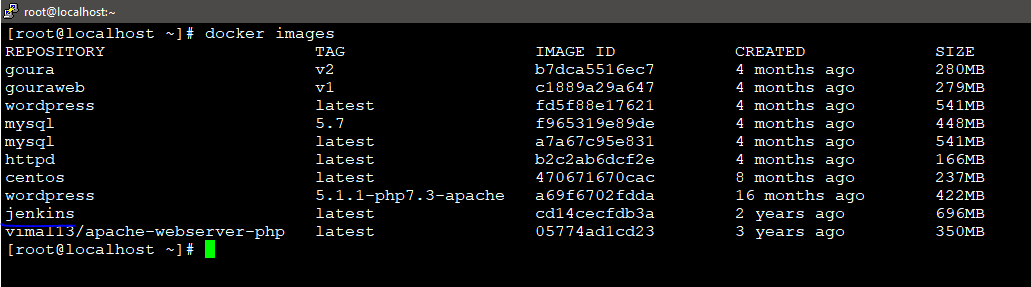
**Step 5** − Now, go to the Linux server and run the following command –

**sudo docker pull jenkins**



*Fig-5*

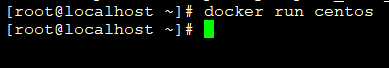
**sudo docker images**



*Fig-6*

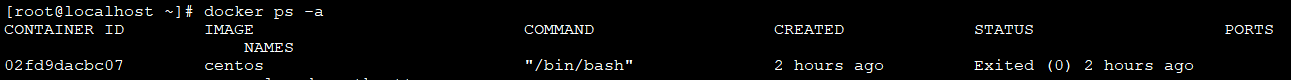
From the above output, you can see that the server has many images: **centos, wordpress,** and **jenkins** etc. Each image has the following attributes −

* **TAG** − This is used to logically tag images.
* **Image ID** − This is used to uniquely identify the image.
* **Created** − The number of days since the image was created.
* **Virtual Size** − The size of the image.

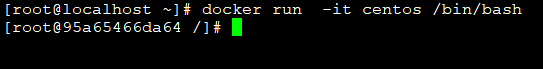


To check the container is running or not, type the following commands.

**docker ps -a**



To enter in to the CentOS, type the following commands.



You will now see the CentOS Docker image downloaded. Now, if we run the Docker **images** command to see the list of images on the system, we should be able to see the **centos** image as well. You can enter in to the CentOS.

## Removing Docker Images

The Docker images on the system can be removed via the **docker rmi** command. Let’s look at this command in more detail.

**docker rmi**

This command is used to remove Docker images.

### Syntax

**docker rmi <ImageID>**

### Options

* **ImageID** − This is the ID of the image which needs to be removed.

### Return Value

The output will provide the Image ID of the deleted Image.

### Example

**sudo docker rmi 470671670cac**

Here, **470671670cac** is the Image ID of the **centos** image. If you will press the above command the it will remove the CentOS image from the host OS.

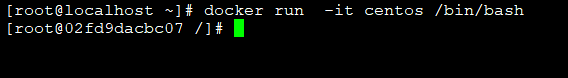
Containers are instances of Docker images that can be run using the Docker run command. The basic purpose of Docker is to run containers. Let’s discuss how to work with containers.

## Running a Container

Running of containers is managed with the Docker **run** command. To run a container in an interactive mode, first launch the Docker container.

**sudo docker run –it centos /bin/bash**

Then type exit and you will return to your host OS shell.



You will then be running in the instance of the CentOS system on the RHEL8 Linux server.

## Displaying Docker Images

To see the list of Docker images on the system, you can issue the following command.

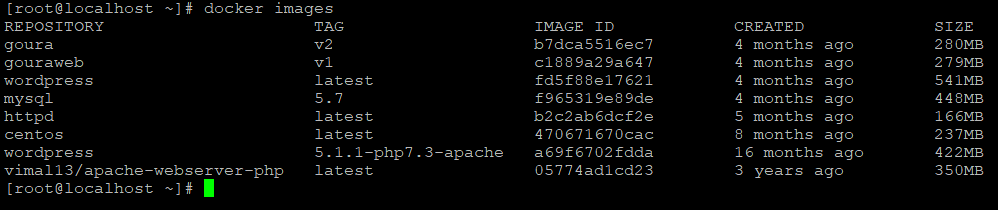
**docker images**

This command is used to display all the images currently installed on the system.

The output will provide the list of images on the system.

### Output

When we run the above command, it will produce the following result –



## docker images -q

This command is used to return only the Image ID’s of the images.

### Syntax

**docker images**

### Options

* **q** − It tells the Docker command to return the Image ID’s only.

### Return Value

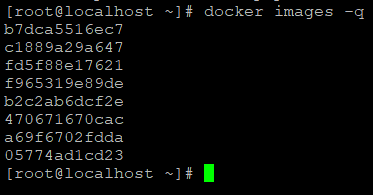
The output will show only the Image ID’s of the images on the Docker host.

### Example

**docker images -q**

### Output

When we run the above command, it will produce the following result −



## docker inspect

This command is used see the details of an image or container.

### Syntax

**docker inspect Repository**

### Options

* **Repository** − This is the name of the Image.

### Return Value

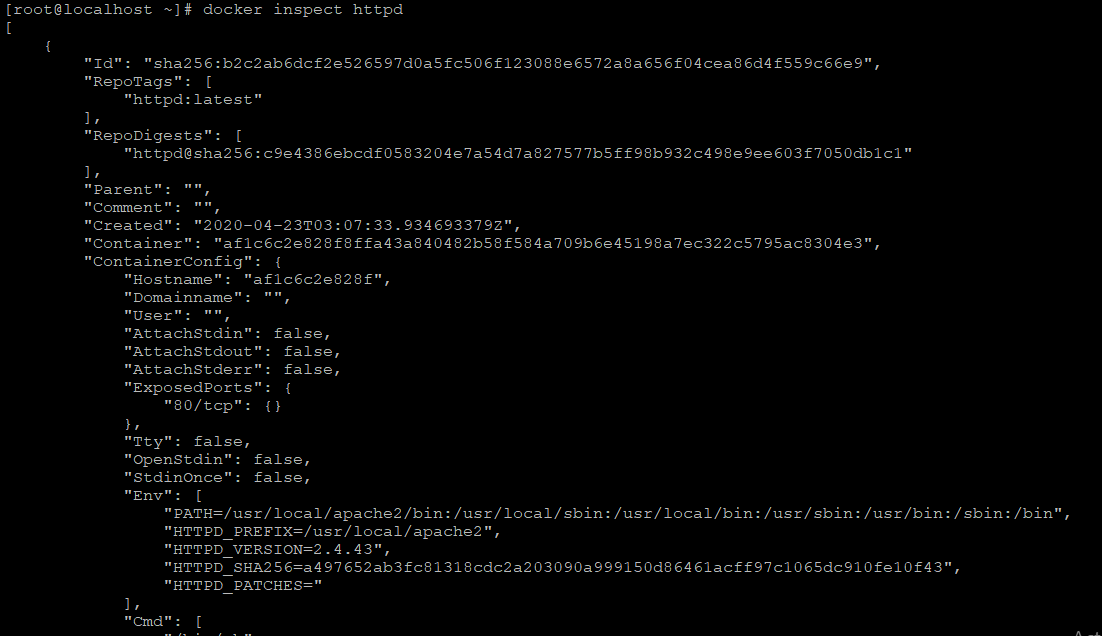
The output will show detailed information on the Image.

### Example

**docker inspect httpd**

### Output

When we run the above command, it will produce the following result −



## Listing of Containers

One can list all of the containers on the machine via the **docker ps** command. This command is used to return the currently running containers.

### Syntax

**docker ps**

### Return Value

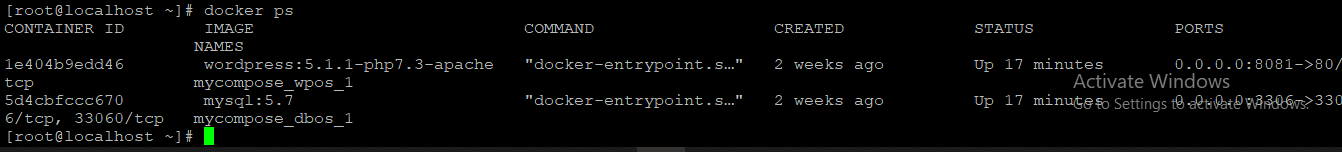
The output will show the currently running containers.

### Example

**docker ps**

### Output

When we run the above command, it will produce the following result −



Let’s see some more variations of the **docker ps** command.

## docker ps -a

This command is used to list all of the containers on the system

### Syntax

**docker ps -a**

### Options

* **─a** − It tells the **docker ps** command to list all of the containers on the system.

### Return Value

The output will show all containers.

### Example

**Docker ps -a**

### Output

When we run the above command, it will produce the following result −



## docker history

With this command, you can see all the commands that were run with an image via a container.

### Syntax

**docker history ImageID**

### Options

* **ImageID** − This is the Image ID for which you want to see all the commands that were run against it.

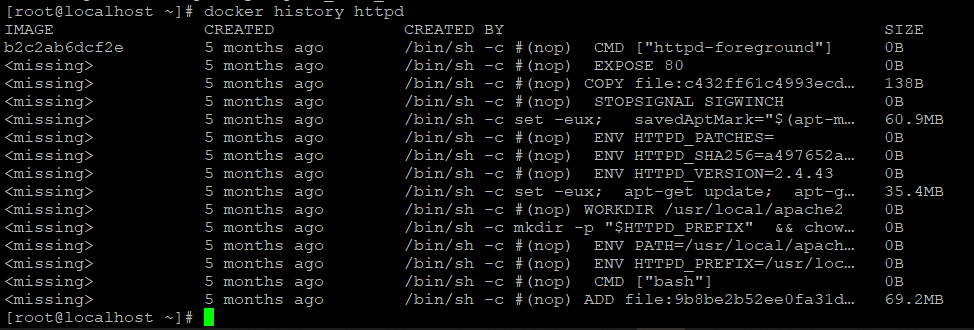
### Example

**Docker history httpd**

The above command will show all the commands that were run against the **centos** image.

### Output

When we run the above command, it will produce the following result −



## docker stop

This command is used to stop a running container.

### Syntax

**docker stop ContainerID**

### Options

* **ContainerID** − This is the Container ID which needs to be stopped.

### Return Value

The output will give the ID of the stopped container.

### Example

**Docker stop 5d4cbfccc670**

The above command will stop the Docker container **5d4cbfccc670**.

### Output

When we run the above command, it will produce the following result −



## docker rm

This command is used to delete a container.

### Syntax

**docker rm ContainerID**

### Options

* **ContainerID** − This is the Container ID which needs to be removed.

### Return Value

The output will give the ID of the removed container.

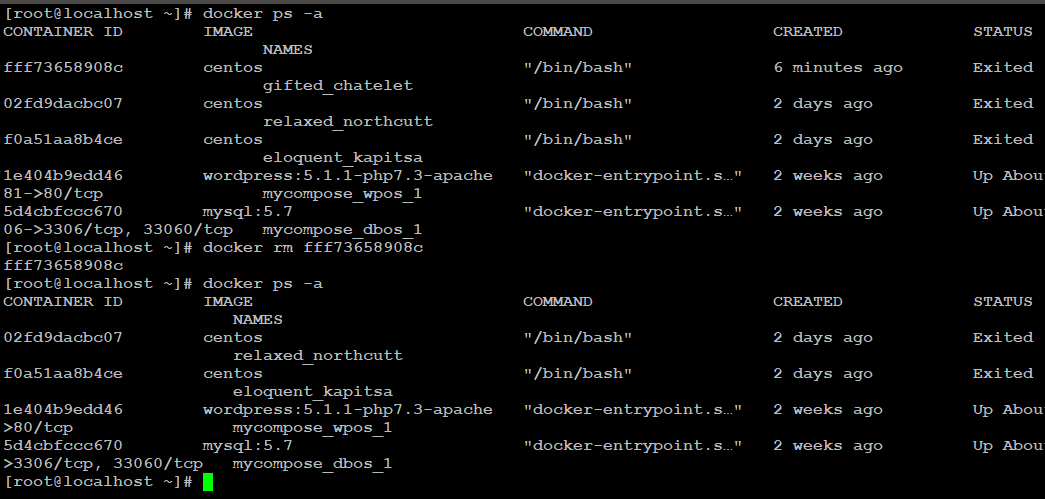
### Example

**Docker rm fff73658908c**

The above command will remove the Docker container **fff73658908c** .

### Output

When we run the above command, it will produce the following result −



## docker stats

This command is used to provide the statistics of a running container.

### Syntax

**docker stats ContainerID**

### Options

* **ContainerID** − This is the Container ID for which the stats need to be provided.

### Return Value

The output will show the CPU and Memory utilization of the Container.

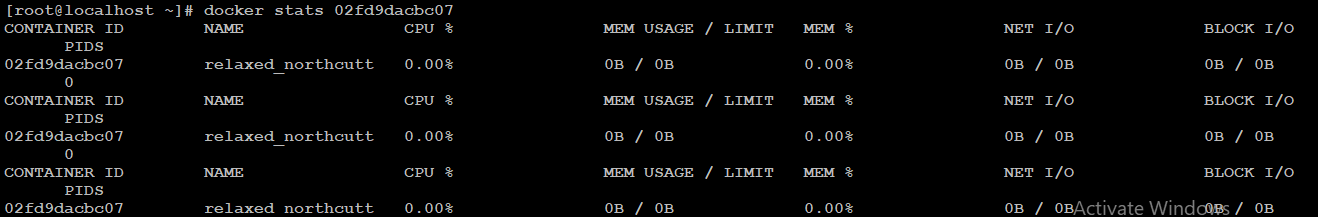
### Example

**Docker stats 02fd9dacbc07**

The above command will provide CPU and memory utilization of the Container **02fd9dacbc07**.

### Output

When we run the above command, it will produce the following result −



## docker attach

This command is used to attach to a running container.

### Syntax

**docker attach ContainerID**

### Options

* **ContainerID** − This is the Container ID to which you need to attach.

### Return Value

None

### Example

**docker attach 02fd9dacbc07**

The above command will attach to the Docker container **02fd9dacbc07**.

### Output

When we run the above command, it will produce the following result −



## docker pause

This command is used to pause the processes in a running container.

### Syntax

**docker pause ContainerID**

### Options

* **ContainerID** − This is the Container ID to which you need to pause the processes in the container.

### Return Value

The ContainerID of the paused container.

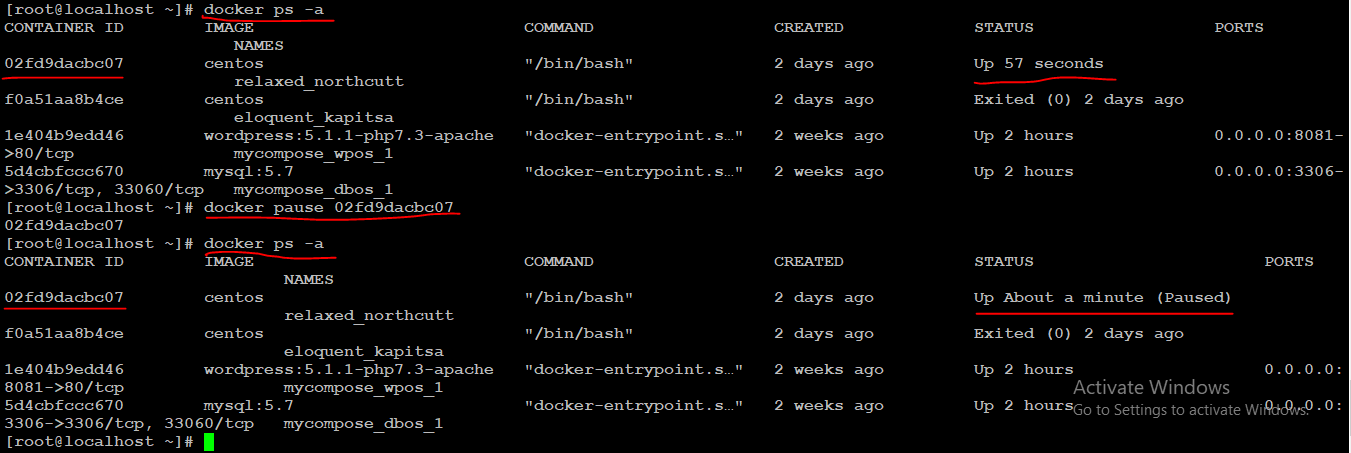
### Example

**Docker pause 02fd9dacbc07**

The above command will pause the processes in a running container **02fd9dacbc07**.

### Output

When we run the above command, it will produce the following result −



## docker unpause

This command is used to **unpause** the processes in a running container.

### Syntax

**docker unpause ContainerID**

### Options

* **ContainerID** − This is the Container ID to which you need to unpause the processes in the container.

### Return Value

The ContainerID of the running container.

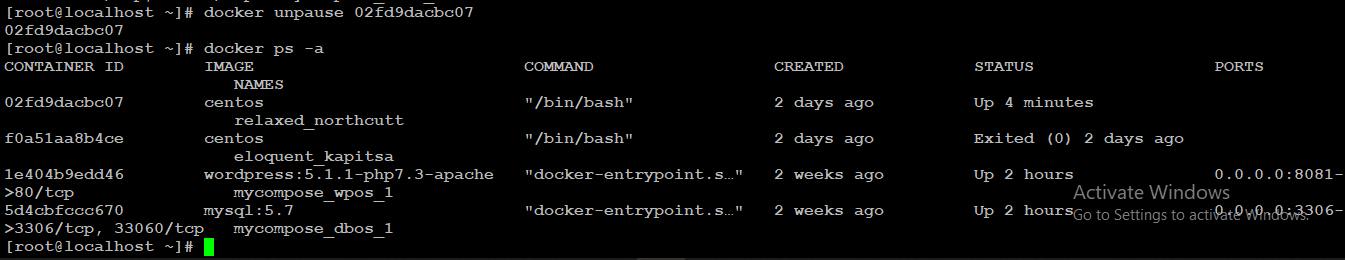
### Example

**Docker unpause 02fd9dacbc07**

The above command will unpause the processes in a running container: **02fd9dacbc07**

### Output

When we run the above command, it will produce the following result −



## docker kill

This command is used to kill the processes in a running container.

### Syntax

**docker kill ContainerID**

### Options

* **ContainerID** − This is the Container ID to which you need to kill the processes in the container.

### Return Value

The ContainerID of the running container.

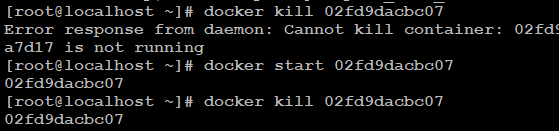
### Example

**docker kill 02fd9dacbc07**

The above command will kill the processes in the running container **02fd9dacbc07**.

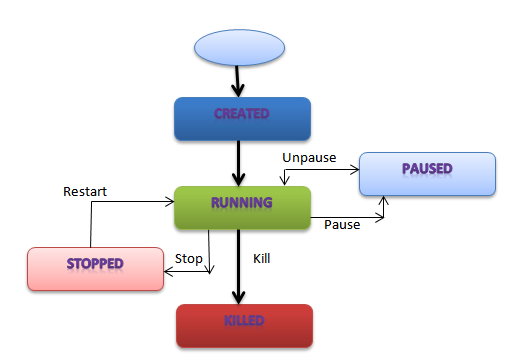
### Output

When we run the above command, it will produce the following result −



## Docker – Container Lifecycle

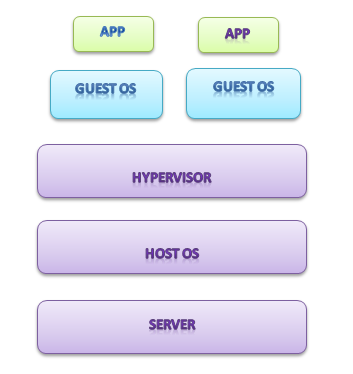
The following illustration explains the entire lifecycle of a Docker container.



* Initially, the Docker container will be in the **created** state.
* Then the Docker container goes into the running state when the Docker **run** command is used.
* The Docker **kill** command is used to kill an existing Docker container.
* The Docker **pause** command is used to pause an existing Docker container.
* The Docker **stop** command is used to pause an existing Docker container.
* The Docker **run** command is used to put a container back from a **stopped** state to a **running** state.

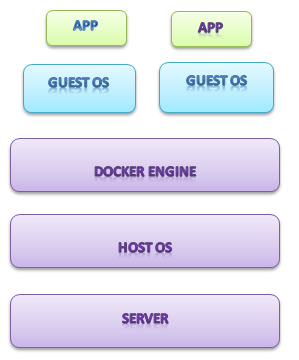
## Docker - Architecture

The following image shows the standard and traditional architecture of **virtualization**.



* The server is the physical server that is used to host multiple virtual machines.
* The Host OS is the base machine such as Linux or Windows.
* The Hypervisor is either VMWare or Windows Hyper V that is used to host virtual machines.
* You would then install multiple operating systems as virtual machines on top of the existing hypervisor as Guest OS.
* You would then host your applications on top of each Guest OS.

The following image shows the new generation of virtualization that is enabled via Dockers. Let’s have a look at the various layers.



The server is the physical server that is used to host multiple virtual machines. So this layer remains the same.

* The Host OS is the base machine such as Linux or Windows. So this layer remains the same.
* Now comes the new generation which is the Docker engine. This is used to run the operating system which earlier used to be virtual machines as Docker containers.
* All of the Apps now run as Docker containers.

The clear advantage in this architecture is that you don’t need to have extra hardware for Guest OS. Everything works as Docker containers.

## Docker - Configuring

## Service/systemctl docker stop

This command is used to stop the Docker **daemon** process.

### Syntax

**service/systemctl docker stop**

### Options

None

### Return Value

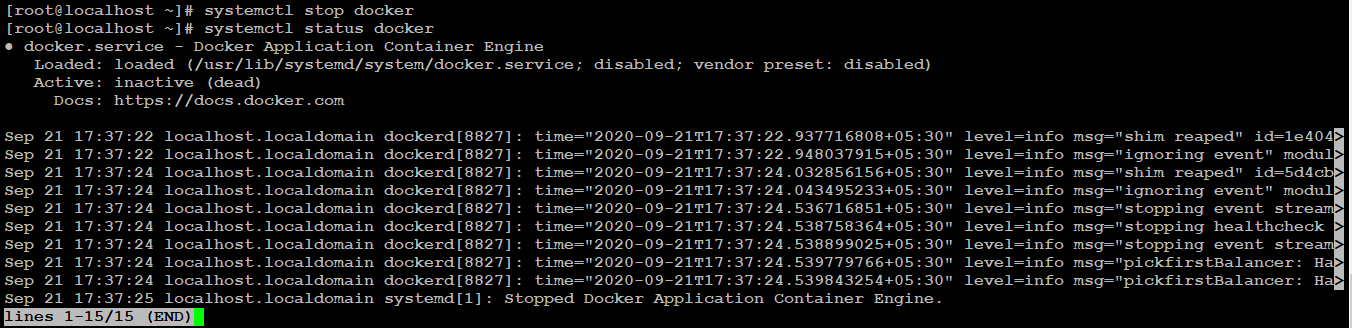
A message showing that the Docker process has stopped.

### Example

**service/systemctl docker stop**

### Output

When we run the above command, it will produce the following result −



## Service/systemctl docker start

This command is used to start the Docker daemon process.

### Syntax

**Service/ systemctl docker start**

### Return Value

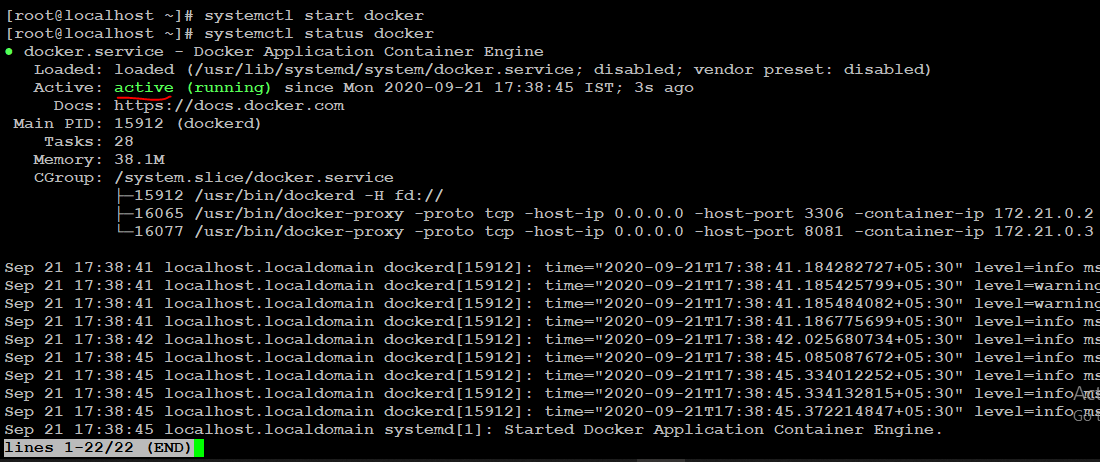
A message showing that the Docker process has started.

### Example

**Service/systemctl docker start**

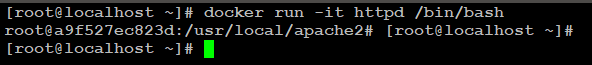
### Output

When we run the above command, it will produce the following result −



## Containers and Shells

When you launch a container, you will also use a **shell command** while launching the container as shown below.

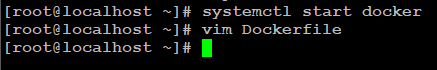


## Docker - File

Docker also gives you the capability to create your own Docker images, and it can be done with the help of **Docker Files**. A Docker File is a simple text file with instructions on how to build your images.

The following steps explain how you should go about creating a Docker File.

**Step 1** − Create a file called **Docker File** and edit it using **vim**. Please note that the name of the file has to be "Dockerfile" with "D" as capital.



**Step 2** − Build your Docker File using the following instructions.

#Creating my first Docker Image

FROM ubuntu

MAINTAINER gourangajena85@gmail.com

RUN apt-get update

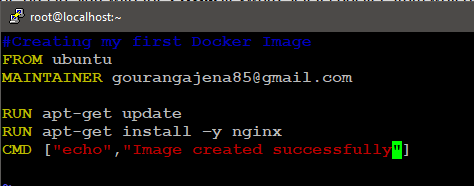
RUN apt-get install –y nginx

CMD [“echo”,”Image created successfully”]

The following points need to be noted about the above file −

* The first line "#This is a sample Image" is a comment. You can add comments to the Docker File with the help of the **#** command
* The next line has to start with the **FROM** keyword. It tells docker, from which base image you want to base your image from. In our example, we are creating an image from the **ubuntu** image.
* The next command is the person who is going to maintain this image. Here you specify the **MAINTAINER** keyword and just mention the email ID.
* The **RUN** command is used to run instructions against the image. In our case, we first update our Ubuntu system and then install the nginx server on our **ubuntu** image.
* The last command is used to display a message to the user.

**Step 3** − Save the file. In the next chapter, we will discuss how to build the image.



## Docker - Building Files

We have created a Docker File in Step-3. Now It’s time to build the Docker File. The Docker File can be built with the following command −

**docker build**

## docker build

This method allows the users to build their own Docker images.

### Syntax

**docker build -t ImageName:TagName dir**

### Options

* **-t** − is to mention a tag to the image
* **ImageName** − This is the name you want to give to your image.
* **TagName** − This is the tag you want to give to your image.
* **Dir** − The directory where the Docker File is present.

### Return Value

None

### Example

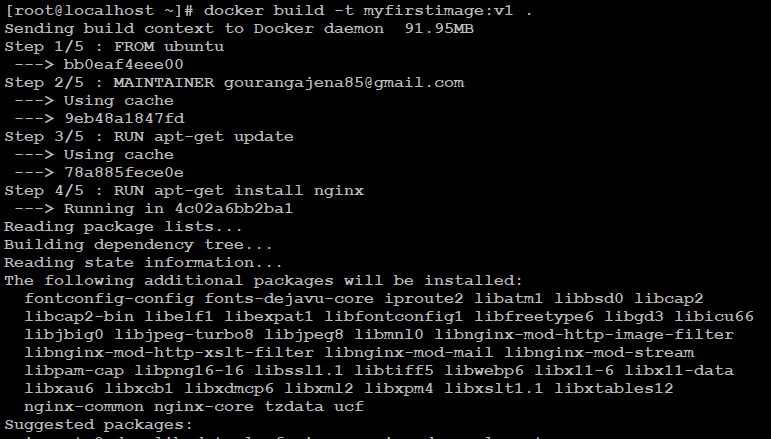
**docker build –t myfirstimage:0.1 .**

Here, **myfirstimage** is the name we are giving to the Image and **0.1** is the tag number we are giving to our image.

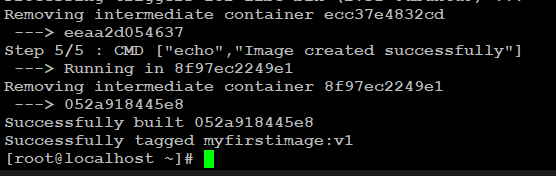
Since the Docker File is in the present working directory, we used "." at the end of the command to signify the present working directory.

### Output

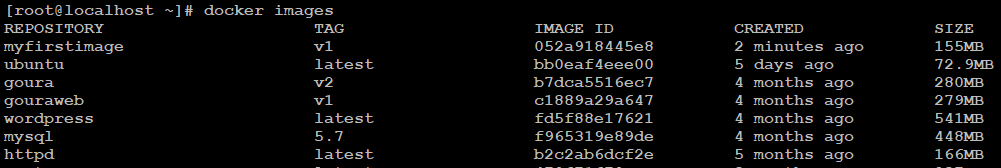
From the output, you will first see that the Ubuntu Image will be downloaded from Docker Hub, because there is no image available locally on the machine.



Finally, when the build is complete, all the necessary commands would have run on the image.



You will then see the successfully built message and the ID of the new Image. When you run the Docker **images command**, you would then be able to see your new image.

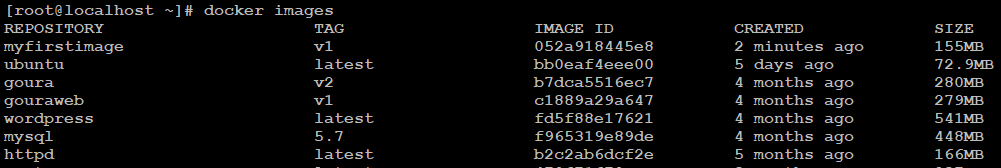


You can now build containers from your new Image.

## Docker - Public Repositories

Public repositories can be used to host Docker images which can be used by everyone else. An example is the images which are available in Docker Hub. Most of the images such as Centos, Ubuntu, and Jenkins are all publicly available for all. We can also make our images available by publishing it to the public repository on Docker Hub.

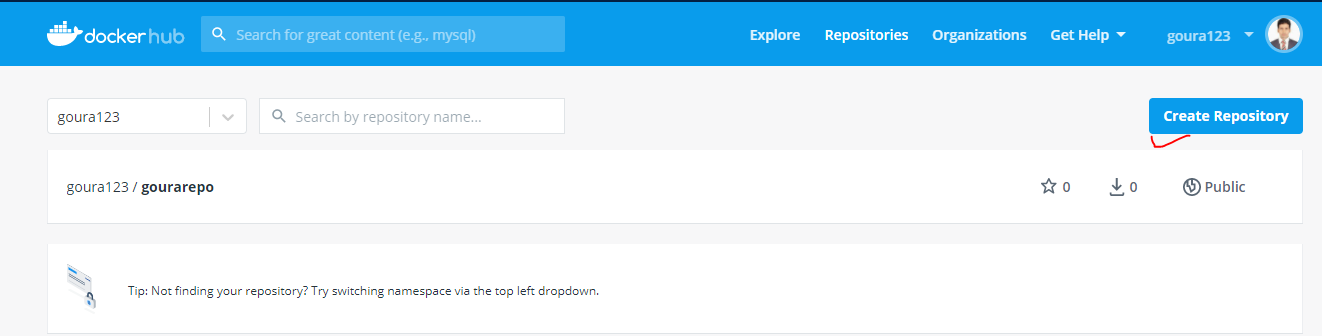
For our example, we will use the **myfirstimage** repository built in the "Building Docker Files" chapter and upload that image to Docker Hub. Let’s first review the images on our Docker host to see what we can push to the Docker registry.



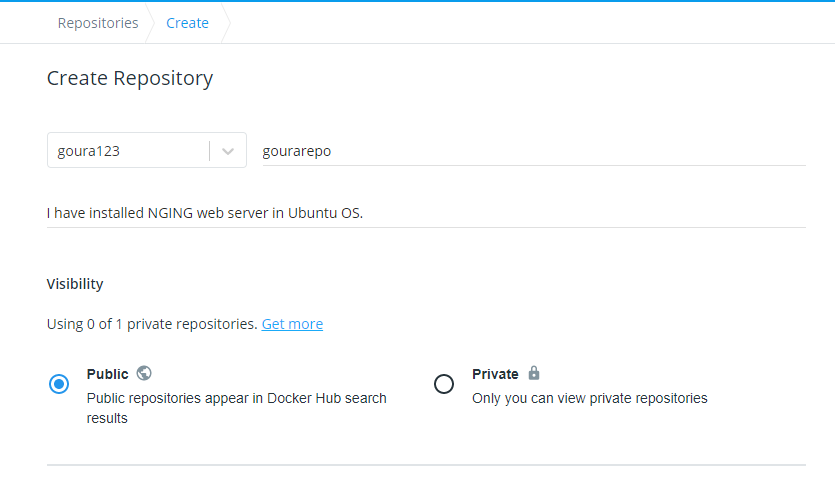
Here, we have our **myfirstimage:v1** image which was created as a part of the “Building Docker Files” chapter. Let’s use this to upload to the Docker public repository.

The following steps explain how you can upload an image to public repository.

**Step 1** − Log into Docker Hub and create your repository. This is the repository where your image will be stored. Go to <https://hub.docker.com/> and log in with your credentials.



**Step 2** − Click the button "Create Repository" on the above screen and create a repository with the name gourarepo. Make sure that the visibility of the repository is public.

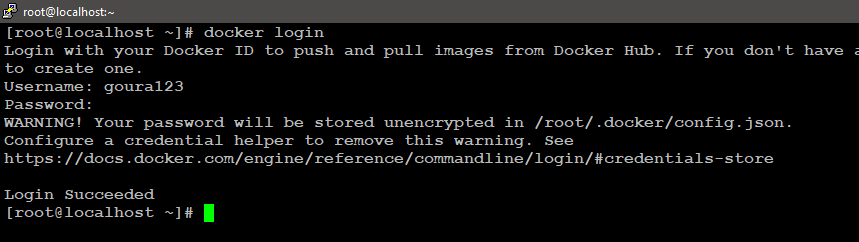


The **pull** command which will be used in our repository is as follows −

**docker pull goura123/gourarepo**

**Step 3** − Now go back to the Docker Host. Here we need to tag our **myfirstimage** to the new repository created in Docker Hub. We can do this via the Docker **tag command**. We will learn more about this **tag command** later in this chapter.

**Step 4** − Issue the Docker login command to login into the Docker Hub repository from the command prompt. The Docker login command will prompt you for the username and password to the Docker Hub repository



**Step 5** − Once the image has been tagged, it’s now time to push the image to the Docker Hub repository. We can do this via the Docker **push** command. We will learn more about this command later in this chapter.

## docker tag

This method allows one to tag an image to the relevant repository.

### Syntax

**docker tag <imageID> <Repositoryname>**

### Options

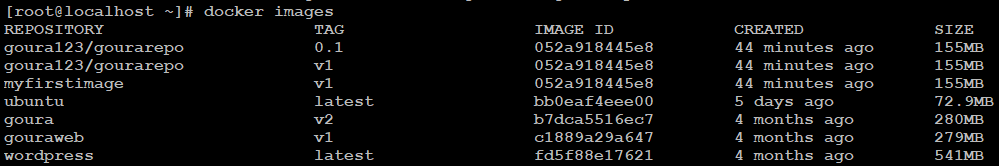
* **imageID** − This is the ImageID which needs to be tagged to the repository.
* **Repositoryname** − This is the repository name to which the ImageID needs to be tagged to.

### Example

**docker tag ag0c1d3744cc goura123/gourarepo:0.1**

### Output

A sample output of the above example is given below.



## docker push

This method allows one to push images to the Docker Hub.

### Syntax

**docker push Repositoryname**

### Repositoryname − This is the repository name which needs to be pushed to the Options

* Docker Hub.

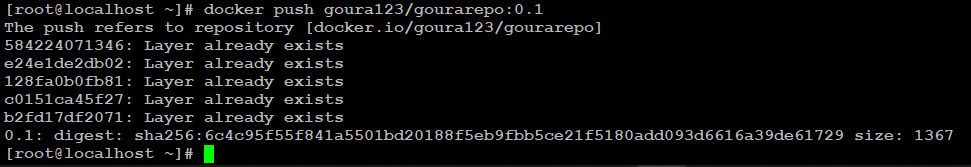
### Return Value

The long ID of the repository pushed to Docker Hub.

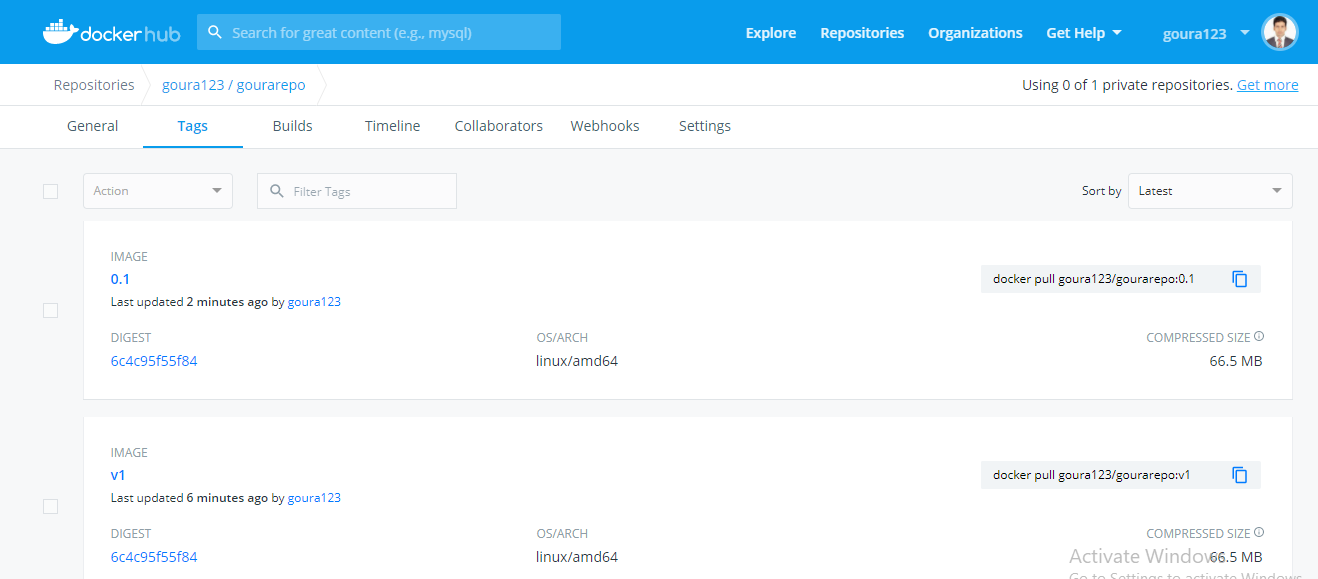
### Example

**docker push goura123/gourarepo:0.1**

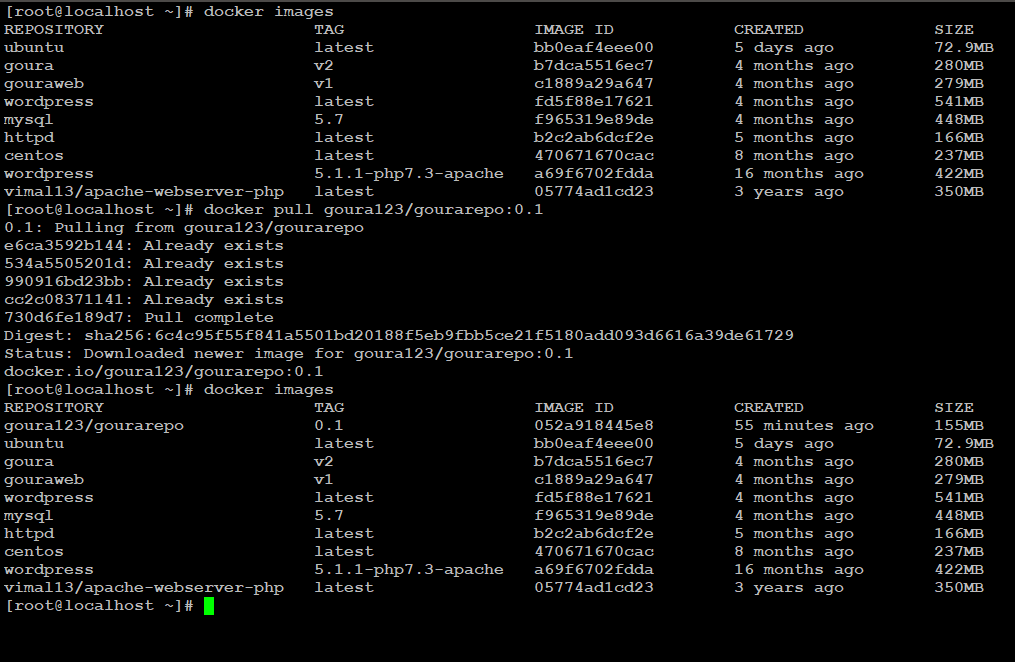
### Output



If you go back to the Docker Hub page and go to your repository, you will see the tag name in the repository.



Now let’s try to pull the repository we uploaded onto our Docker host. Let’s first delete the images, **myfirstimage:0.1** and **goura123/gourarepo:0.1**, from the local Docker host. Let’s use the Docker **pull command** to pull the repository from the Docker Hub.



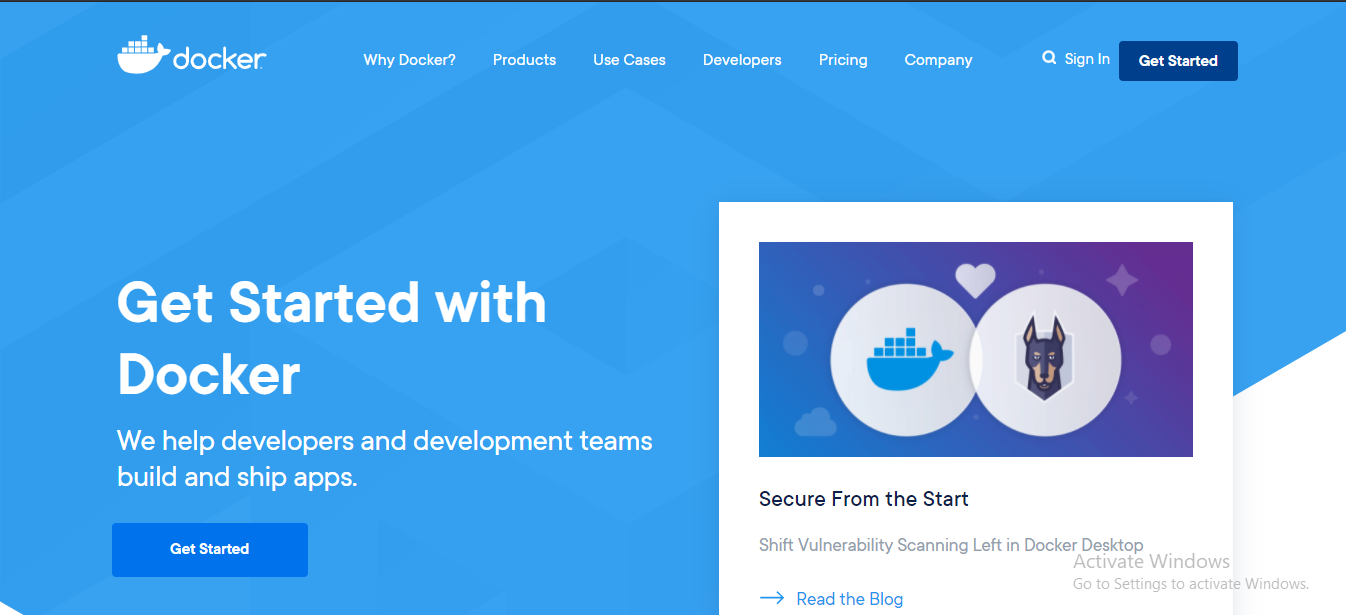
From the above screenshot, you can see that the Docker **pull** command has taken our new repository from the Docker Hub and placed it on our machine.

## Docker - Managing Ports

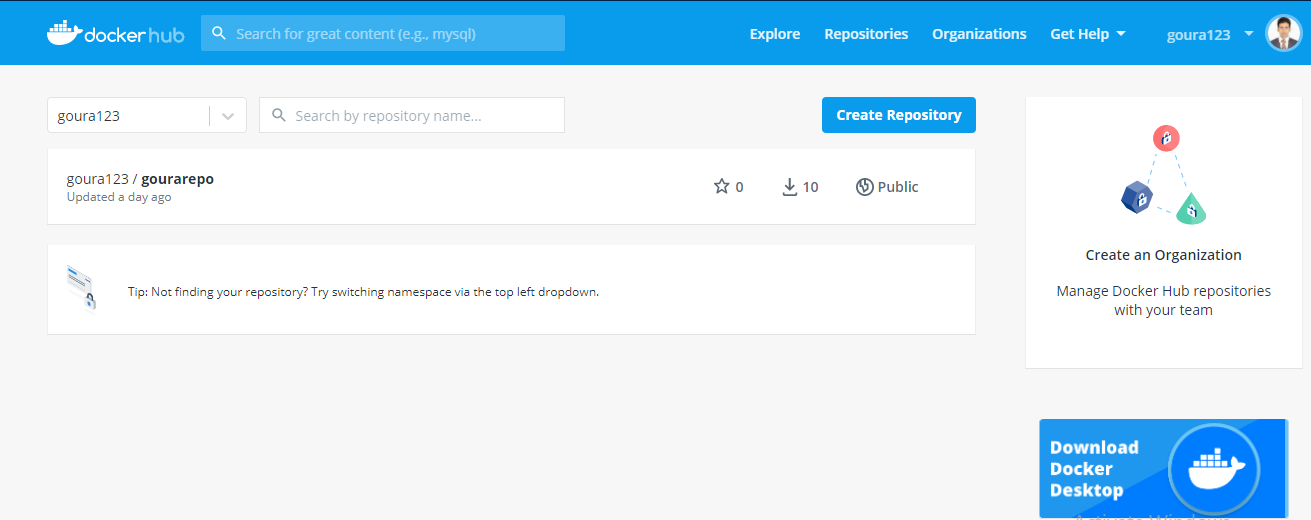
In Docker, the containers themselves can have applications running on ports. When you run a container, if you want to access the application in the container via a port number, you need to map the port number of the container to the port number of the Docker host. Let’s look at an example of how this can be achieved.

In our example, we are going to download the Jenkins container from Docker Hub. We are then going to map the Jenkins port number to the port number on the Docker host.

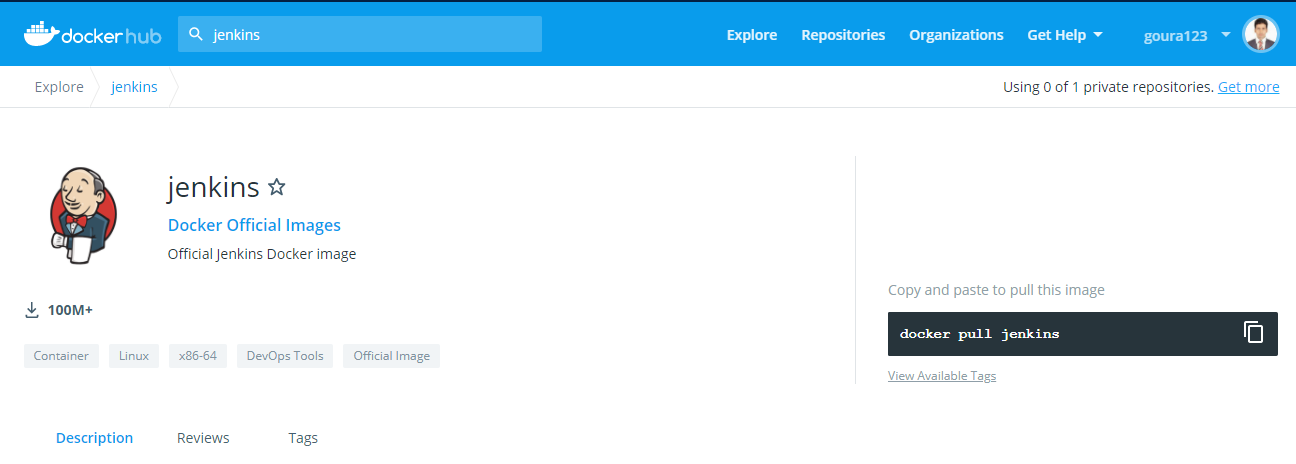
**Step 1** − First, you need to do a simple sign-up on Docker Hub.



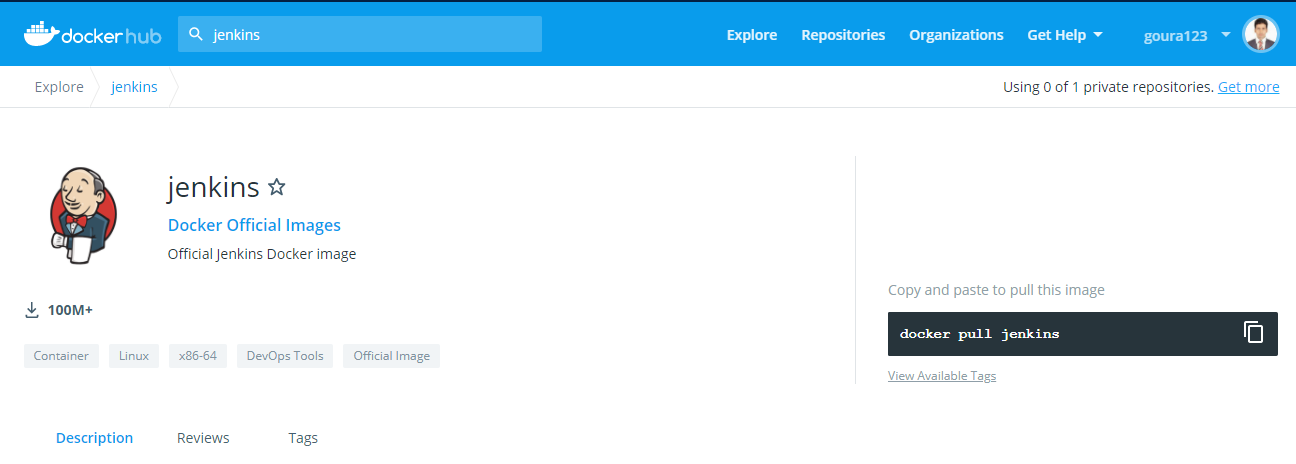
**Step 2** − Once you have signed up, you will be logged into Docker Hub.



**Step 3** − Next, let’s browse and find the Jenkins image.

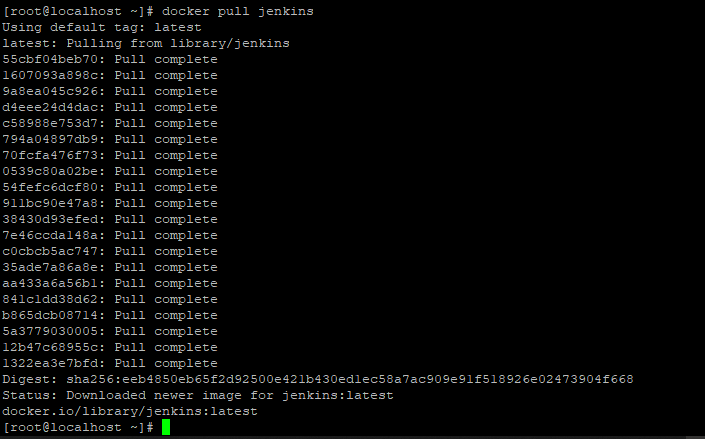


**Step 4** − you can see the Docker **pull** command in the same page in the right side. This will be used to download the Jenkins Image onto the local RHEL server.



**Step 5** − Now go to the Ubuntu server and run the command −

**sudo docker pull jenkins**



**Step 6** − To understand what ports are exposed by the container, you should use the Docker **inspect command** to inspect the image.

## docker inspect

This method allows one to return low-level information on the container or image.

### Syntax

**docker inspect Container/Image**

### Options

* **Container/Image** − The container or image to inspect

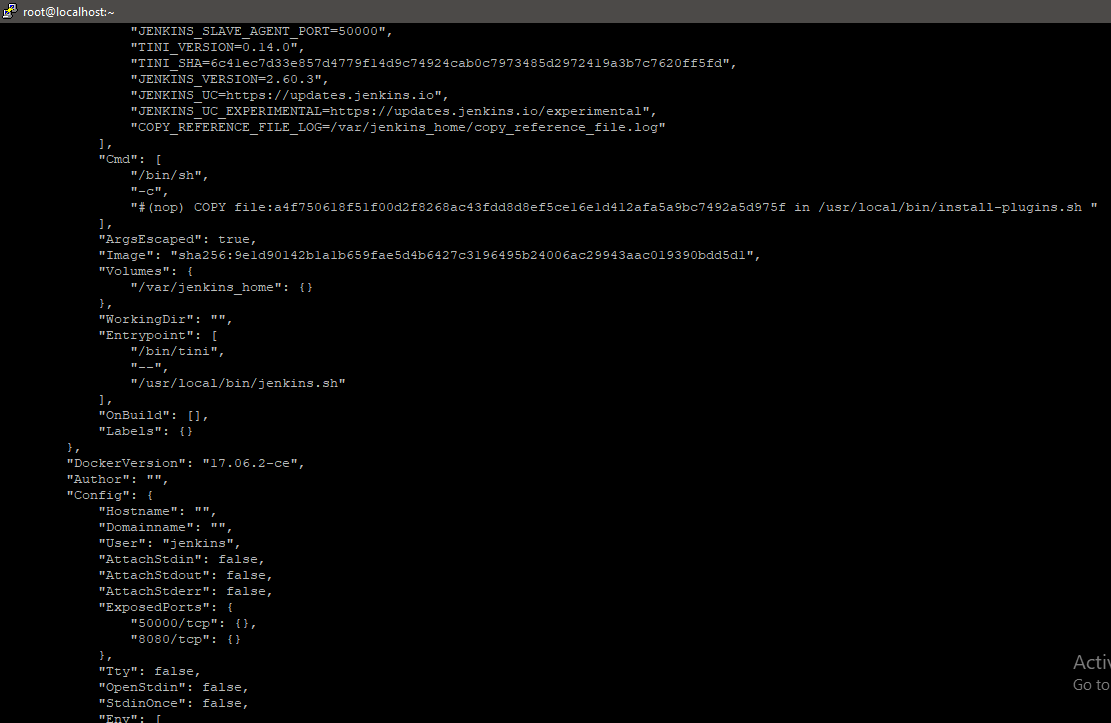
### Return Value

The low-level information of the image or container in JSON format.

### Example

**docker inspect jenkins**

### Output



The output of the **inspect** command gives a JSON output. If we observe the output, we can see that there is a section of "ExposedPorts" and see that there are two ports mentioned. One is the **data port** of 8080 and the other is the **control port** of 50000.

To run Jenkins and map the ports, you need to change the Docker **run** command and add the ‘p’ option which specifies the port mapping. So, you need to run the following command −

**sudo docker run -p 8080:8080 -p 50000:50000 jenkins**

The left-hand side of the port number mapping is the Docker host port to map to and the right-hand side is the Docker container port number.

When you open the browser and navigate to the Docker host on port 8080, you will see Jenkins up and running.

