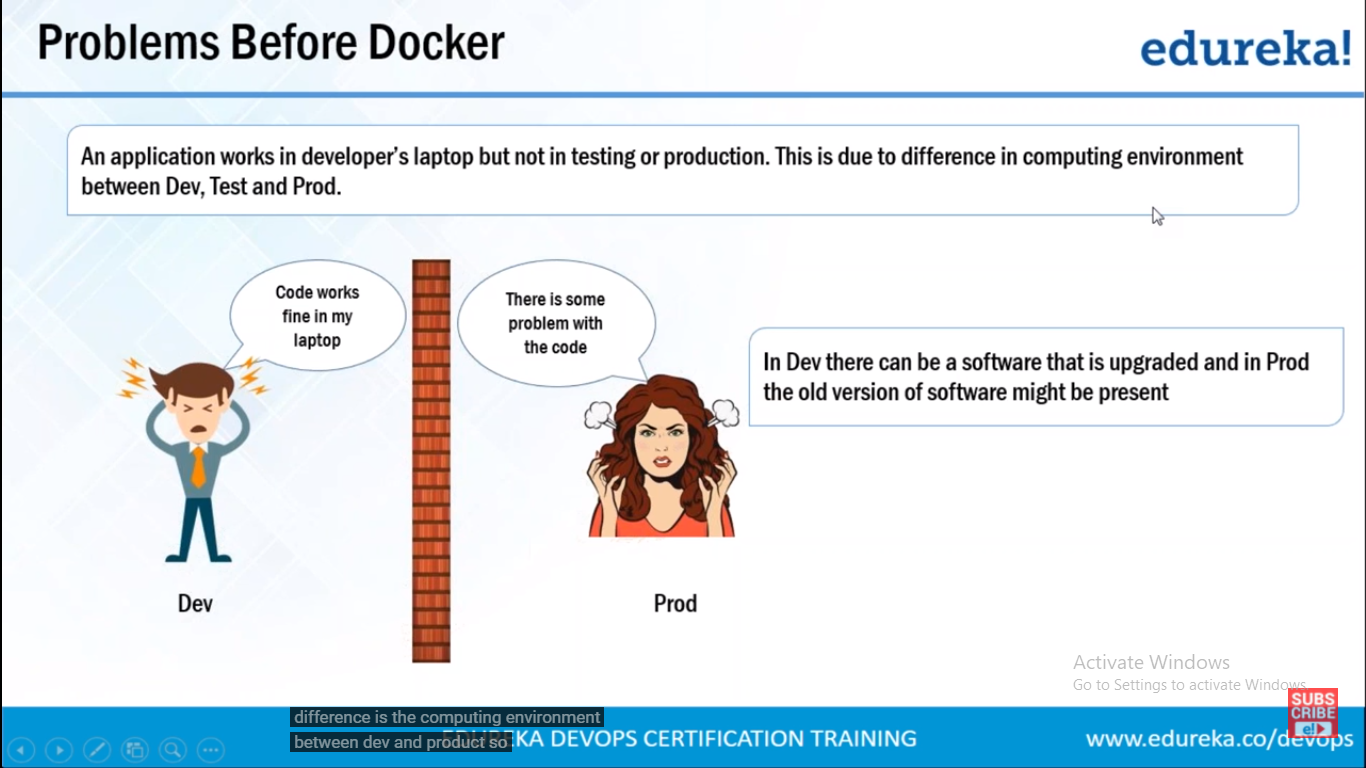
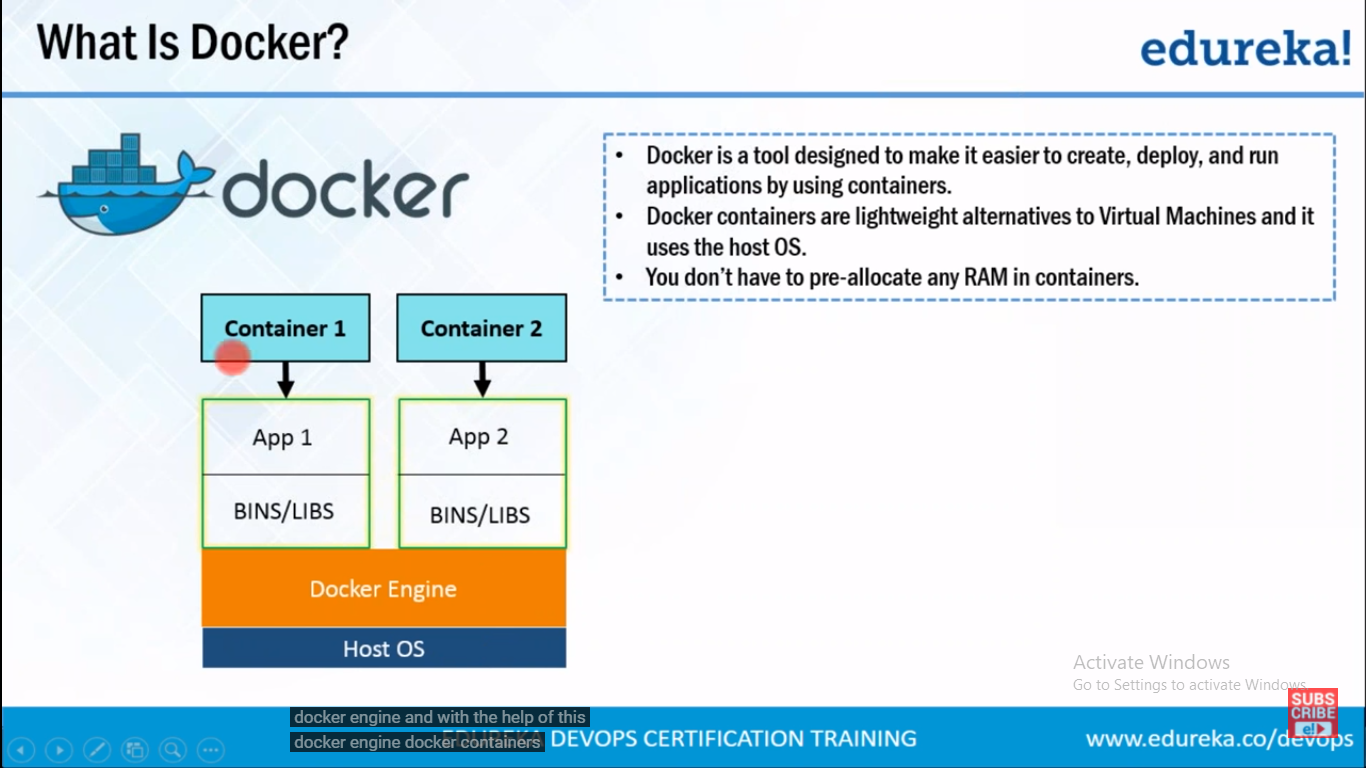
**https://www.tutorialspoint.com/docker/docker\_working\_with\_containers.htm**

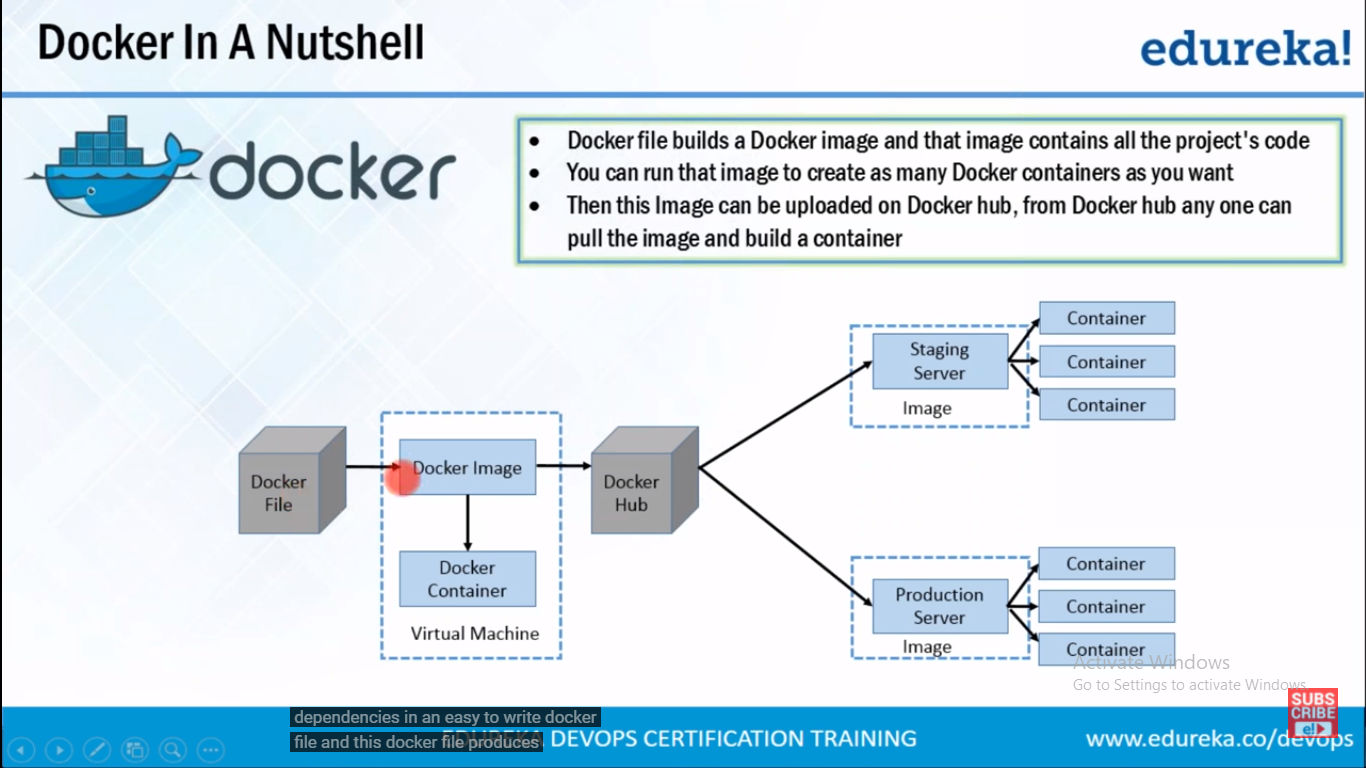
Why do we need docker…?



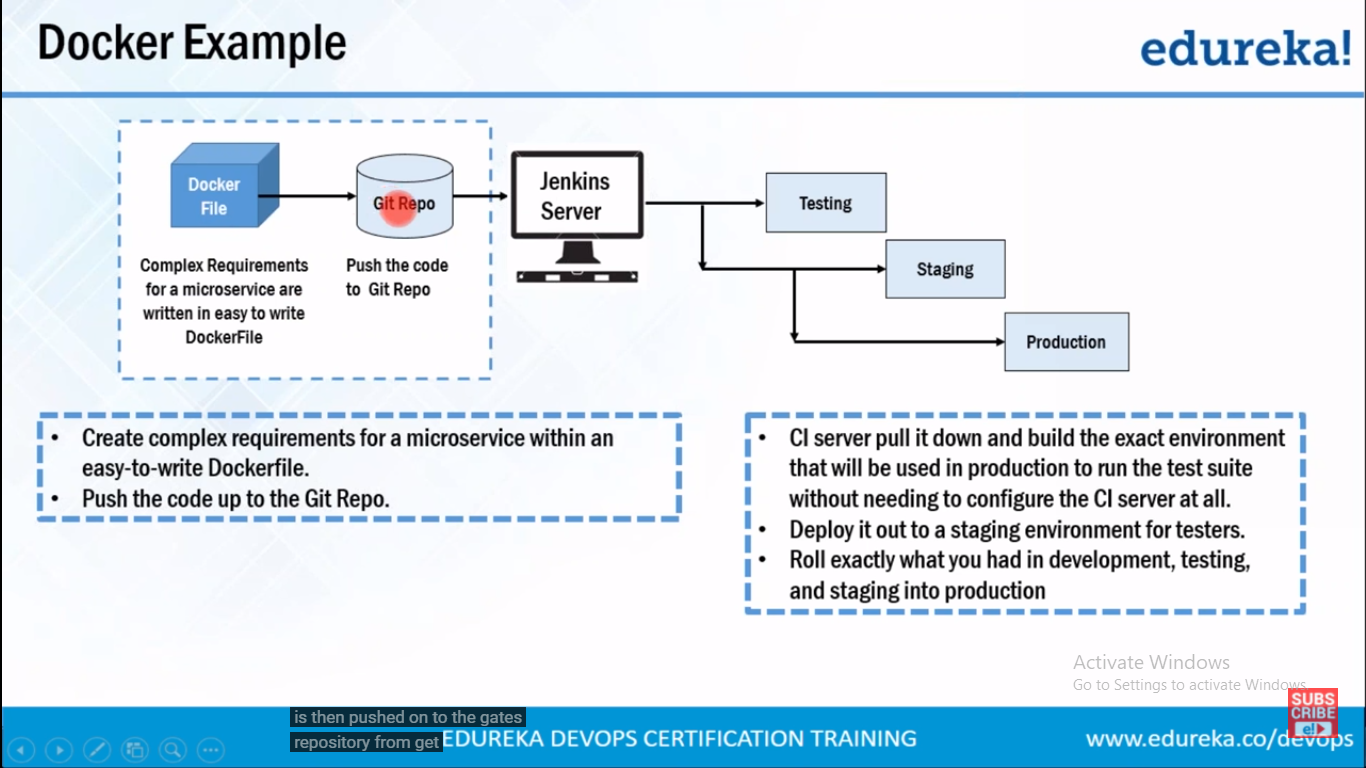
Above issue will sove when we use docker---How

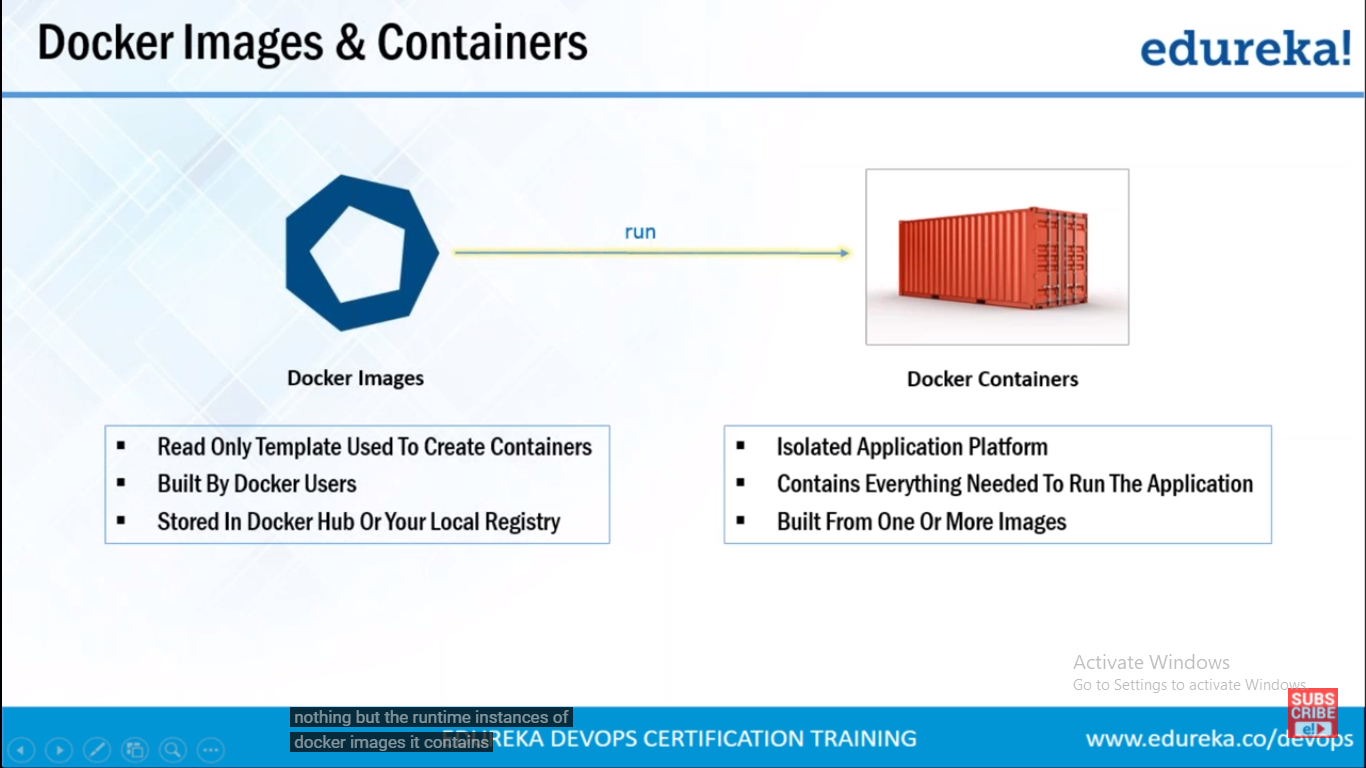
Because will be having same environment throughout the software delivery lifecycle, dev test and production





Docker hub is git repository for docker images… it contains public repo, u can pull images fro tht, u can upload ur own images as well..

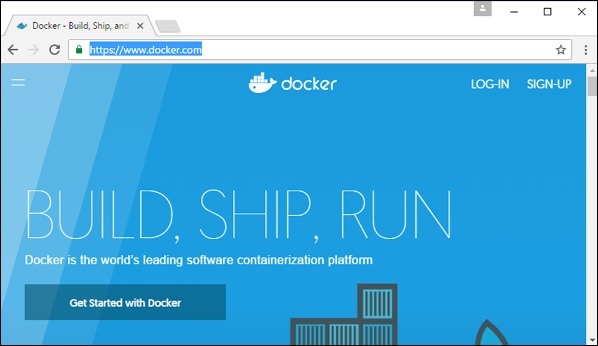




# **Docker - Overview**

Docker is a container management service. The keywords of Docker are **develop, ship** and **run** anywhere. The whole idea of Docker is for developers to easily develop applications, ship them into containers which can then be deployed anywhere.

The initial release of Docker was in March 2013 and since then, it has become the buzzword for modern world development, especially in the face of Agile-based projects.



Features of Docker

* Docker has the ability to reduce the size of development by providing a smaller footprint of the operating system via containers.
* With containers, it becomes easier for teams across different units, such as development, QA and Operations to work seamlessly across applications.
* You can deploy Docker containers anywhere, on any physical and virtual machines and even on the cloud.
* Since Docker containers are pretty lightweight, they are very easily scalable.

Components of Docker

Docker has the following components

* **Docker for Mac** − It allows one to run Docker containers on the Mac OS.
* **Docker for Linux** − It allows one to run Docker containers on the Linux OS.
* **Docker for Windows** − It allows one to run Docker containers on the Windows OS.
* **Docker Engine** − It is used for building Docker images and creating Docker containers.
* **Docker Hub** − This is the registry which is used to host various Docker images.
* **Docker Compose** − This is used to define applications using multiple Docker containers.

We will discuss all these components in detail in the subsequent chapters.

The official site for Docker is <https://www.docker.com/> The site has all information and documentation about the Docker software. It also has the download links for various operating systems.

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.

# **Docker - Images**

In Docker, everything is based on Images. An image is a combination of a file system and parameters. Let’s take an example of the following command in Docker.

docker run hello-world

* The Docker command is specific and tells the Docker program on the Operating System that something needs to be done.
* The **run** command is used to mention that we want to create an instance of an image, which is then called a **container**.
* Finally, "hello-world" represents the image from which the container is made.

Now let’s look at how we can use the CentOS image available in Docker Hub to run CentOS on our Ubuntu machine. We can do this by executing the following command on our Ubuntu machine −

sudo docker run -it centos /bin/bash

Note the following points about the above **sudo** command −

* We are using the **sudo** command to ensure that it runs with **root** access.
* Here, **centos** is the name of the image we want to download from Docker Hub and install on our Ubuntu machine.
* **─it** is used to mention that we want to run in **interactive mode**.
* **/bin/bash** is used to run the bash shell once CentOS is up and running.

## 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.

### **Syntax**

docker images

### **Options**

None

### **Return Value**

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

### **Example**

sudo docker images

### **Output**

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



From the above output, you can see that the server has three images: **centos, newcentos,** and **jenkins**. 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.

## Downloading Docker Images

Images can be downloaded from Docker Hub using the Docker **run** command. Let’s see in detail how we can do this.

### **Syntax**

The following syntax is used to run a command in a Docker container.

docker run image

### **Options**

* **Image** − This is the name of the image which is used to run the container.

### **Return Value**

The output will run the command in the desired container.

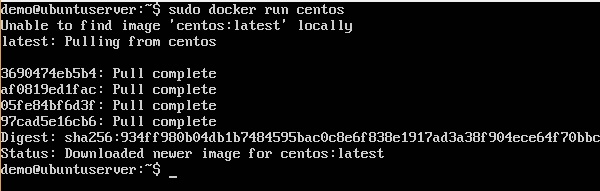
### **Example**

sudo docker run centos

This command will download the **centos** image, if it is not already present, and run the OS as a container.

### **Output**

When we run the above command, we will get the following result −



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.



## 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.

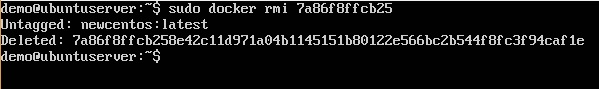
### **Example**

sudo docker rmi 7a86f8ffcb25

Here, **7a86f8ffcb25** is the Image ID of the **newcentos** image.

### **Output**

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



Let’s see some more Docker commands on images.

## 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**

sudo 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**

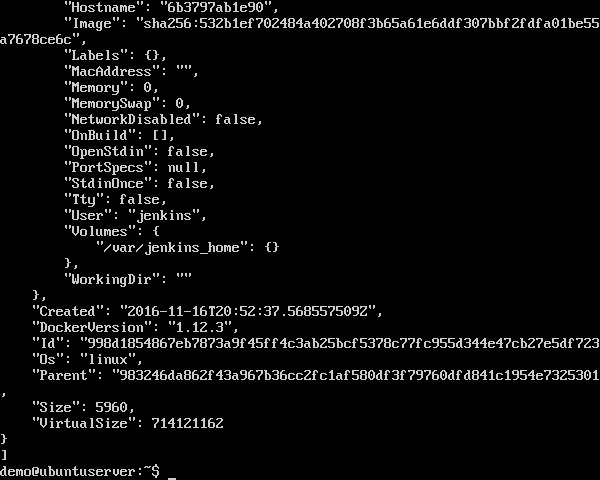
The output will show detailed information on the Image.

### **Example**

sudo docker inspect jenkins

### **Output**

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



# **Docker - Containers**

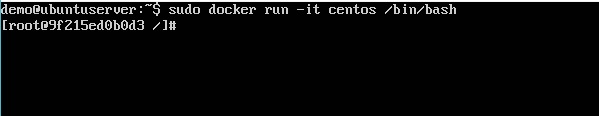
Containers are instances of Docker images that can be run using the Docker run command. The basic purpose of Docker is to run containers. Let’s discuss how to work with containers.

## 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 hit Crtl+p and you will return to your OS shell.



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

## 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.

docker ps

### **Syntax**

docker ps

### **Options**

None

### **Return Value**

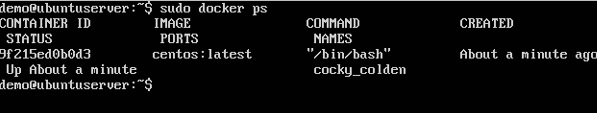
The output will show the currently running containers.

### **Example**

sudo 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**

sudo 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.

### **Return Value**

The output will show all the commands run against that image.

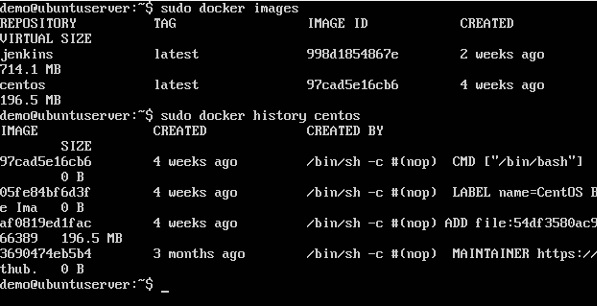
### **Example**

sudo docker history centos

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 - Working with Containers**

In this chapter, we will explore in detail what we can do with containers.

## docker top

With this command, you can see the top processes within a container.

### **Syntax**

docker top ContainerID

### **Options**

* **ContainerID** − This is the Container ID for which you want to see the top processes.

### **Return Value**

The output will show the top-level processes within a container.

### **Example**

sudo docker top 9f215ed0b0d3

The above command will show the top-level processes within a container.

### **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.

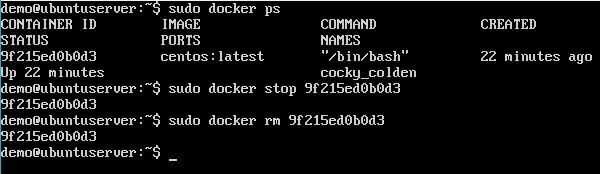
### **Example**

sudo docker stop 9f215ed0b0d3

The above command will stop the Docker container **9f215ed0b0d3**.

### **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.

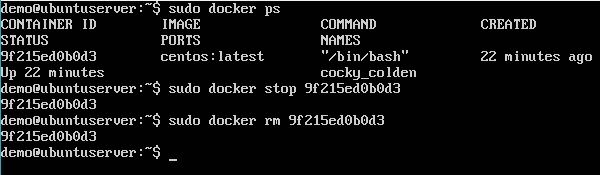
### **Example**

sudo docker rm 9f215ed0b0d3

The above command will remove the Docker container **9f215ed0b0d3**.

### **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.

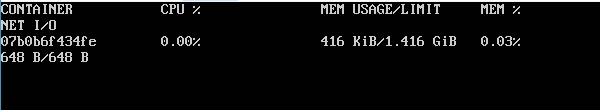
### **Example**

sudo docker stats 9f215ed0b0d3

The above command will provide CPU and memory utilization of the Container **9f215ed0b0d3**.

### **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**

sudo docker attach 07b0b6f434fe

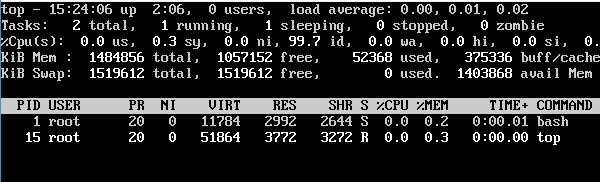
The above command will attach to the Docker container **07b0b6f434fe**.

### **Output**

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



Once you have attached to the Docker container, you can run the above command to see the process utilization in that Docker container.



## 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**

sudo docker pause 07b0b6f434fe

The above command will pause the processes in a running container **07b0b6f434fe**.

### **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.

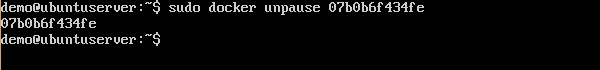
### **Example**

sudo docker unpause 07b0b6f434fe

The above command will unpause the processes in a running container: 07b0b6f434fe

### **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.

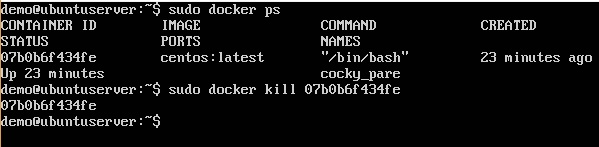
### **Example**

sudo docker kill 07b0b6f434fe

The above command will kill the processes in the running container **07b0b6f434fe**.

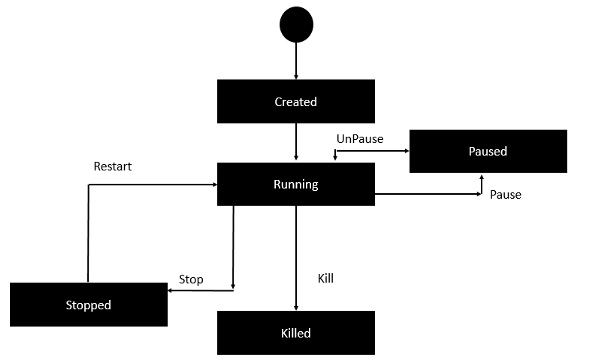
### **Output**

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



## Docker – Container Lifecycle

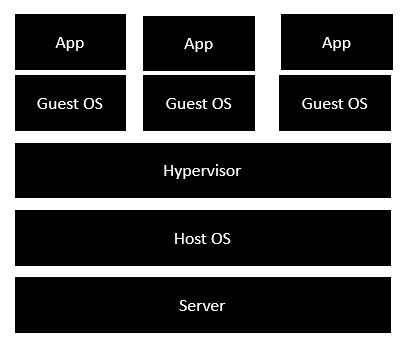
The following illustration explains the entire lifecycle of a Docker container.



* 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.

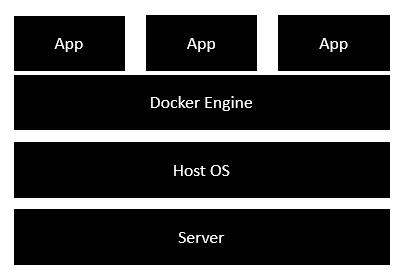
# **Docker - Architecture**

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**

In this chapter, we will look at the different options to configure Docker.

## service docker stop

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

### **Syntax**

service docker stop

### **Options**

None

### **Return Value**

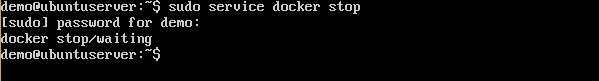
A message showing that the Docker process has stopped.

### **Example**

sudo service docker stop

### **Output**

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



## service docker start

This command is used to start the Docker daemon process.

### **Syntax**

service docker start

### **Options**

None

### **Return Value**

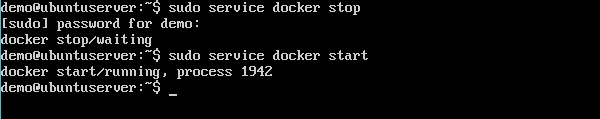
A message showing that the Docker process has started.

### **Example**

sudo service docker start

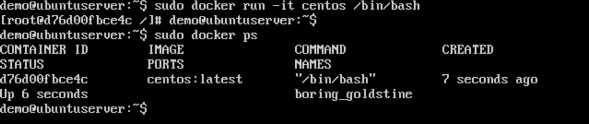
### **Output**

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



# **Docker - Containers and Shells**

By default, when you launch a container, you will also use a **shell command** while launching the container as shown below. This is what we have seen in the earlier chapters when we were working with containers.



In the above screenshot, you can observe that we have issued the following command −

sudo docker run –it centos /bin/bash

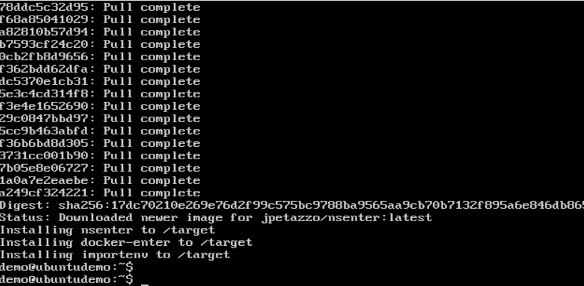
We used this command to create a new container and then used the Ctrl+P+Q command to exit out of the container. It ensures that the container still exists even after we exit from the container.

We can verify that the container still exists with the Docker **ps** command. If we had to exit out of the container directly, then the container itself would be destroyed.

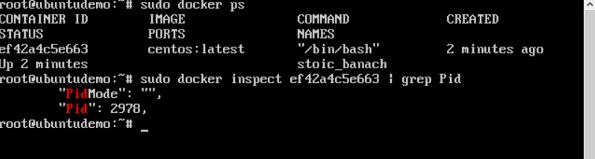
Now there is an easier way to attach to containers and exit them cleanly without the need of destroying them. One way of achieving this is by using the **nsenter** command.

Before we run the **nsenter** command, you need to first install the **nsenter** image. It can be done by using the following command −

docker run --rm -v /usr/local/bin:/target jpetazzo/nsenter



Before we use the **nsenter** command, we need to get the Process ID of the container, because this is required by the **nsenter** command. We can get the Process ID via the Docker **inspect command** and filtering it via the **Pid**.



As seen in the above screenshot, we have first used the **docker ps** command to see the running containers. We can see that there is one running container with the ID of ef42a4c5e663.

We then use the Docker **inspect** command to inspect the configuration of this container and then use the **grep** command to just filter the Process ID. And from the output, we can see that the Process ID is 2978.

Now that we have the process ID, we can proceed forward and use the **nsenter** command to attach to the Docker container.

## nsenter

This method allows one to attach to a container without exiting the container.

### **Syntax**

nsenter –m –u –n –p –i –t containerID command

### **Options**

* **-u** is used to mention the **Uts namespace**
* **-m** is used to mention the **mount namespace**
* **-n** is used to mention the **network namespace**
* **-p** is used to mention the **process namespace**
* **-i** s to make the container run in interactive mode.
* **-t** is used to connect the I/O streams of the container to the host OS.
* **containerID** − This is the ID of the container.
* **Command** − This is the command to run within the container.

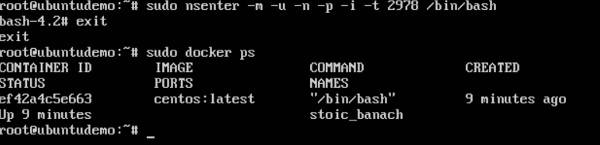
### **Return Value**

None

### **Example**

sudo nsenter –m –u –n –p –i –t 2978 /bin/bash

### **Output**



From the output, we can observe the following points −

* The prompt changes to the **bash shell** directly when we issue the **nsenter** command.
* We then issue the **exit** command. Now normally if you did not use the **nsenter** command, the container would be destroyed. But you would notice that when we run the **nsenter** command, the container is still up and running.

# **Docker - File**

In the earlier chapters, we have seen the various Image files such as Centos which get downloaded from **Docker hub** from which you can spin up containers. An example is again shown below.



If we use the Docker **images** command, we can see the existing images in our system. From the above screenshot, we can see that there are two images: **centos** and **nsenter**.

But 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.

#This is a sample Image

FROM ubuntu

MAINTAINER demousr@gmail.com

RUN apt-get update

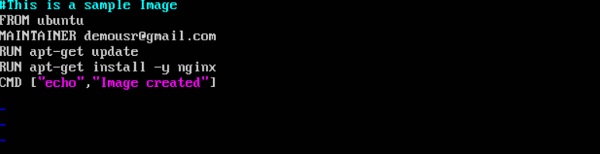
RUN apt-get install –y nginx

CMD [“echo”,”Image created”]

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 created our Docker File in the last chapter. It’s now time to build the Docker File. The Docker File can be built with the following command −

docker build

Let’s learn more about this command.

## 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**

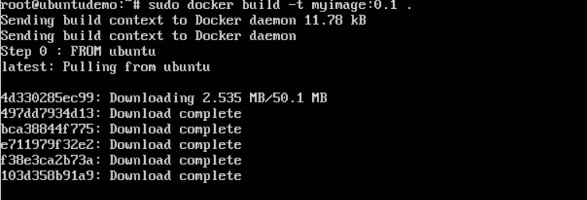
sudo docker build –t myimage:0.1.

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

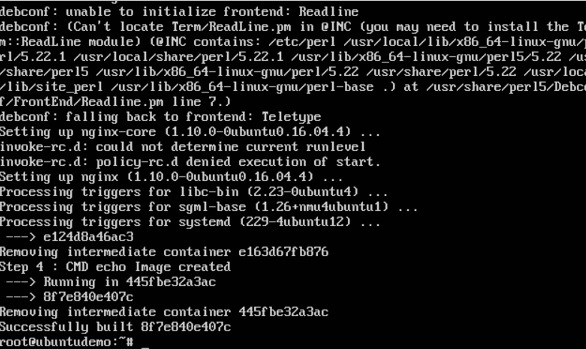
Since the Docker File is in the present working directory, we used "." at the end of the command to signify the present working directory.

### **Output**

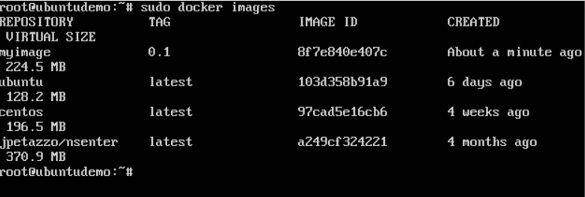
From the output, you will first see that the Ubuntu Image will be downloaded from Docker Hub, because there is no image available locally on the machine.



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.

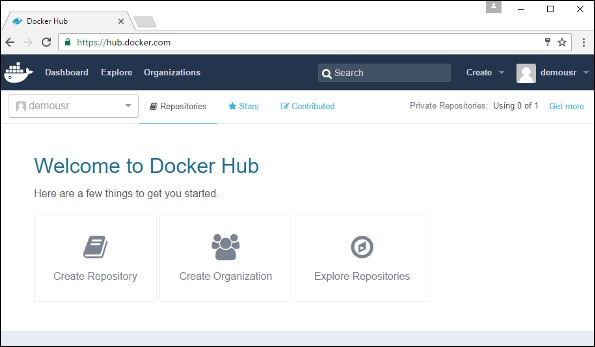
For our example, we will use the **myimage** repository built in the "Building Docker Files" chapter and upload that image to Docker Hub. Let’s first review the images on our Docker host to see what we can push to the Docker registry.



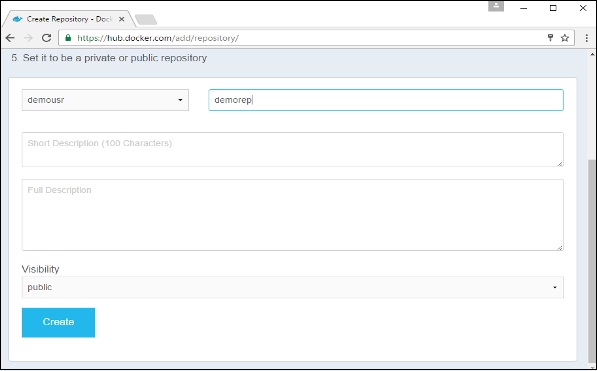
Here, we have our **myimage:0.1** image which was created as a part of the “Building Docker Files” chapter. Let’s use this to upload to the Docker public repository.

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

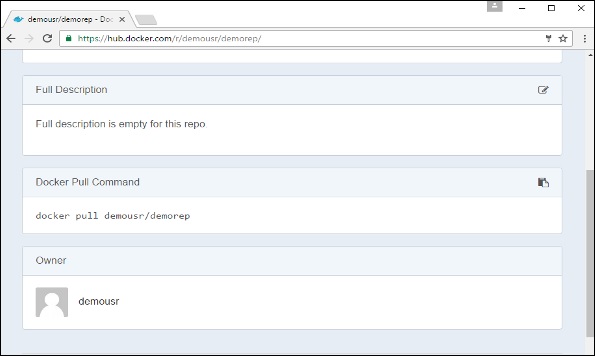
**Step 1** − Log into Docker Hub and create your repository. This is the repository where your image will be stored. Go to <https://hub.docker.com/> and log in with your credentials.



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



Once the repository is created, make a note of the **pull** command which is attached to the repository.



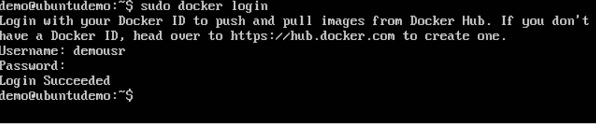
The **pull** command which will be used in our repository is as follows −

docker pull demousr/demorep

**Step 3** − Now go back to the Docker Host. Here we need to tag our **myimage** 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.

### **Return Value**

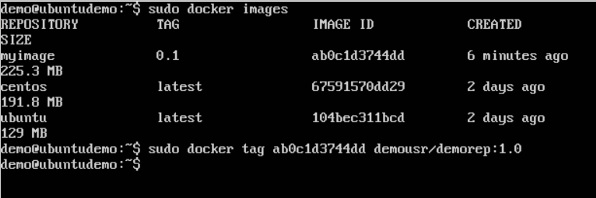
None

### **Example**

sudo docker tag ab0c1d3744dd demousr/demorep:1.0

### **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

### **Options**

* **Repositoryname** − This is the repository name which needs to be pushed to the Docker Hub.

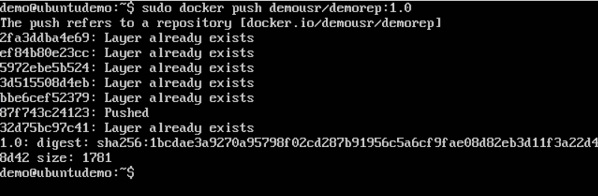
### **Return Value**

The long ID of the repository pushed to Docker Hub.

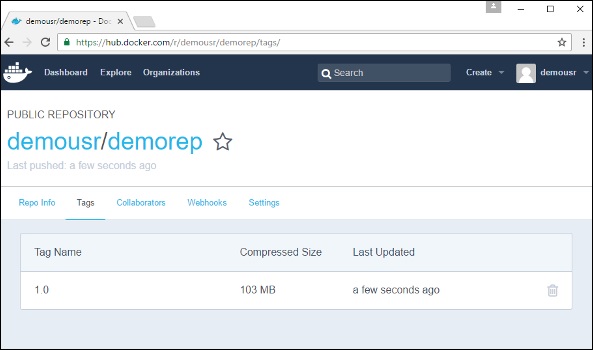
### **Example**

sudo docker push demousr/demorep:1.0

### **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, **myimage:0.1** and **demousr/demorep:1.0**, 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.