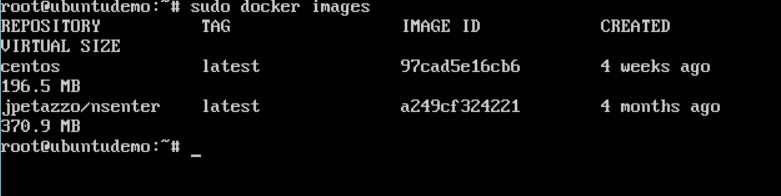
**Docker – Docker File**

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 nameof 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

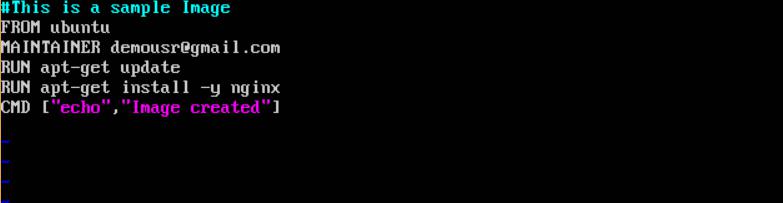
RUN apt-get install –y nginx

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.



**Building Docker**

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**

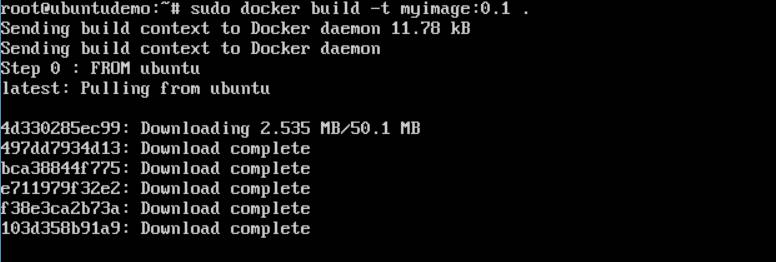
sudo docker build –t myimage:0.1 .

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

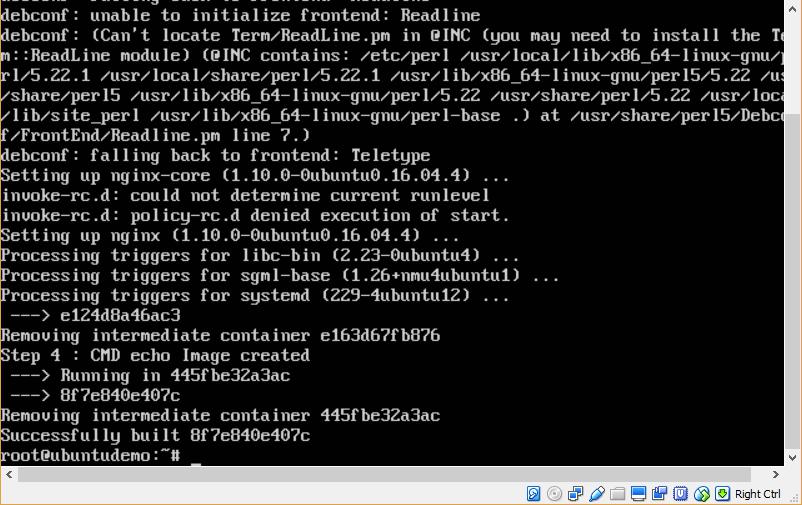
Since the Docker File is in the present working directory, we used "**.**" at the end of the command to signify the present working directory.

**Output**

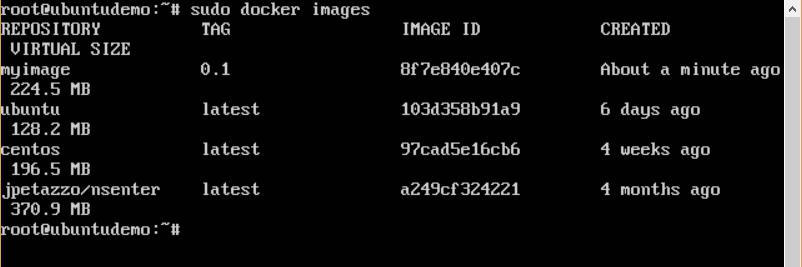
From the output, you will first see that the Ubuntu Image will be downloaded from Docker Hub, because there is no image available locally on the machine.



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.

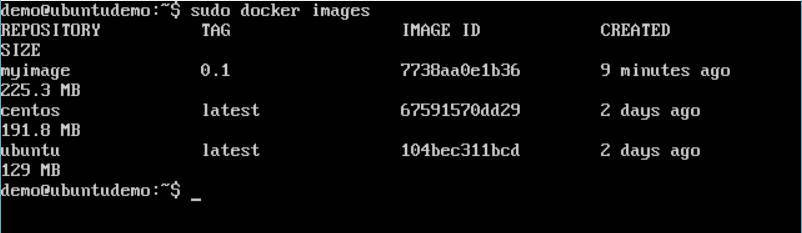


You can now build containers from your new Image.

**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.

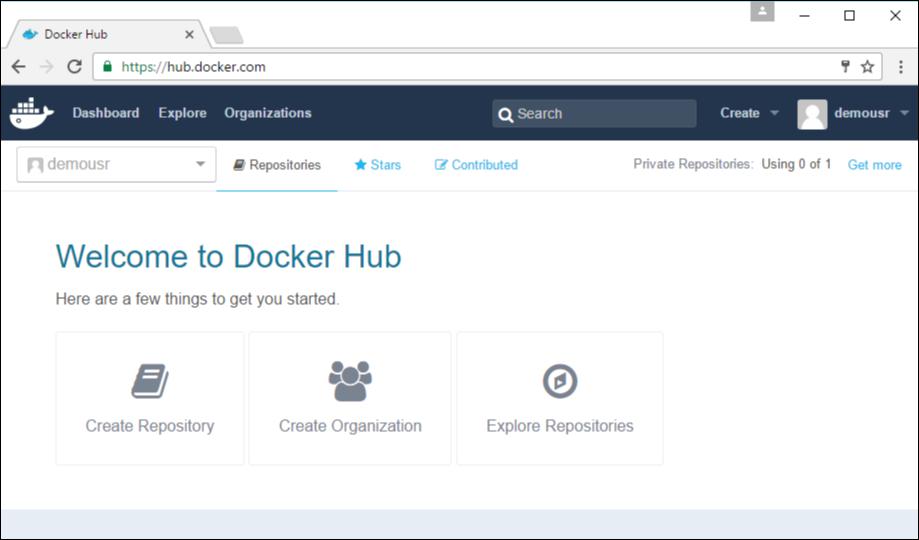
For our example, we will use the **myimage** repository built in the "Building Docker Files" chapter and upload that image to Docker Hub. Let’s first review the images on our Docker host to see what we can push to the Docker registry.



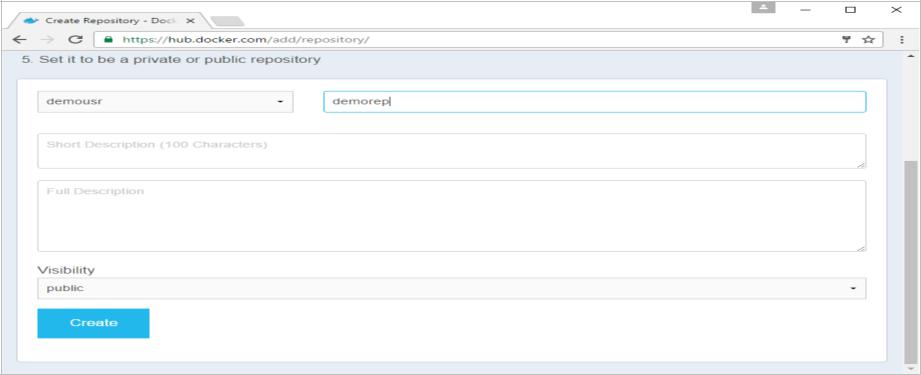
Here, we have our **myimage:0.1** image which was created as a part of the “Building Docker Files” chapter. Let’s use this to upload to the Docker public repository.

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

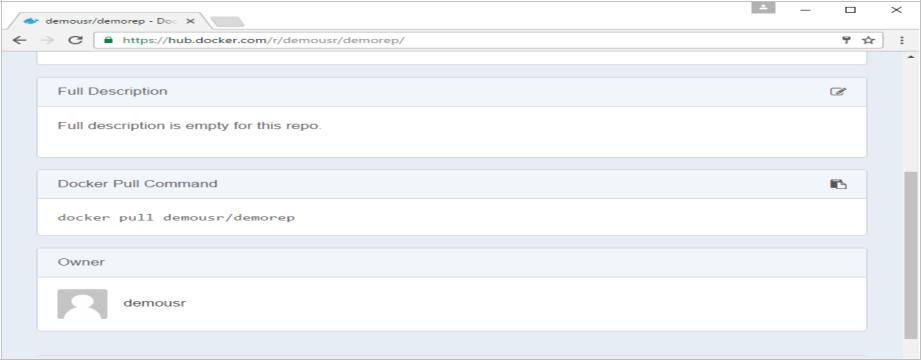
**Step 1**: Log into Docker Hub and create your repository. This is the repository where yourimage will be stored. Go to <https://hub.docker.com/>and log in with your credentials.



**Step 2**: Click the button "Create Repository" on the above screen and create a repositorywith 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.



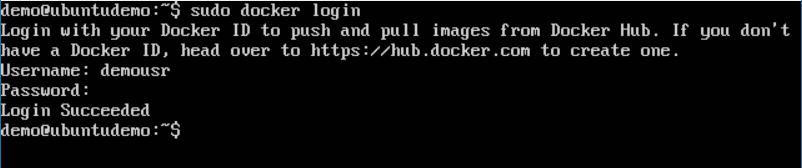
The **pull** command which will be used in our repository is as follows:

docker pull demousr/demorep

**Step 3**: Now go back to the Docker Host. Here we need to tag our **myimage** to the newrepository 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 thecommand 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 theDockerHub 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 betagged to.

**Return Value**

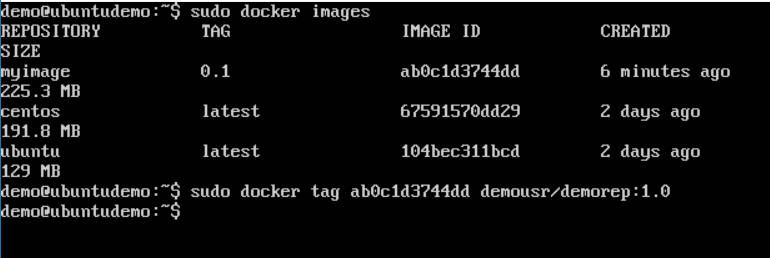
None

**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 theDocker Hub.

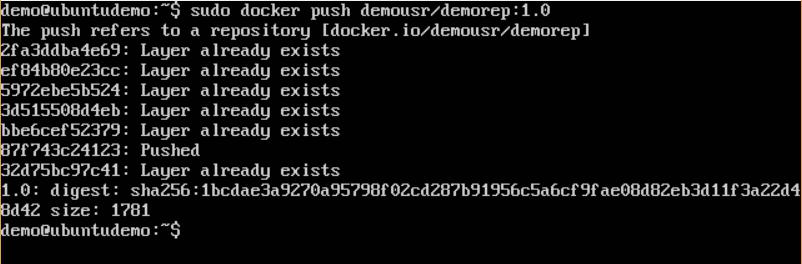
**Return Value**

The long ID of the repository pushed to Docker Hub.

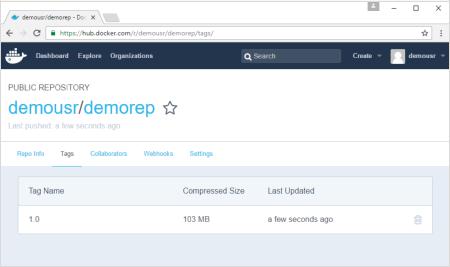
**Example**

sudo docker push demousr/demorep:1.0

**Output**



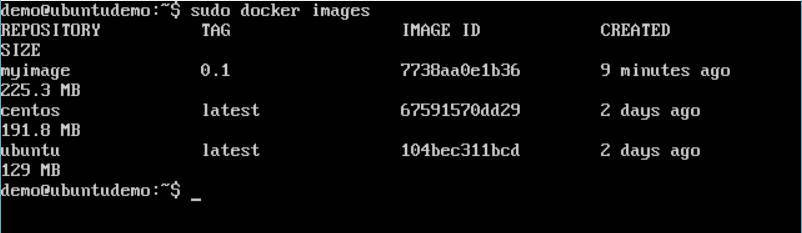
If you go back to the Docker Hub page and go to your repository, you will see the tag name in the repository.



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.

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.

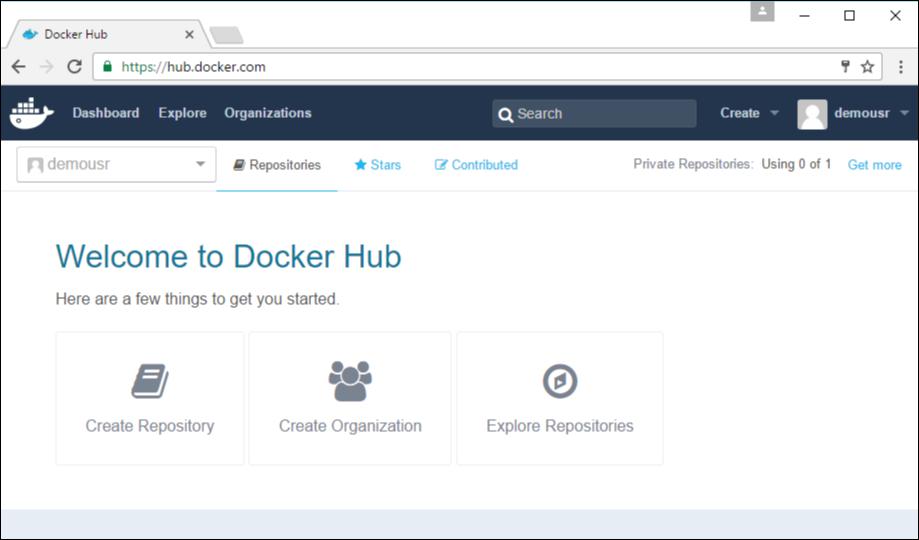
For our example, we will use the **myimage** repository built in the "Building Docker Files" chapter and upload that image to Docker Hub. Let’s first review the images on our Docker host to see what we can push to the Docker registry.



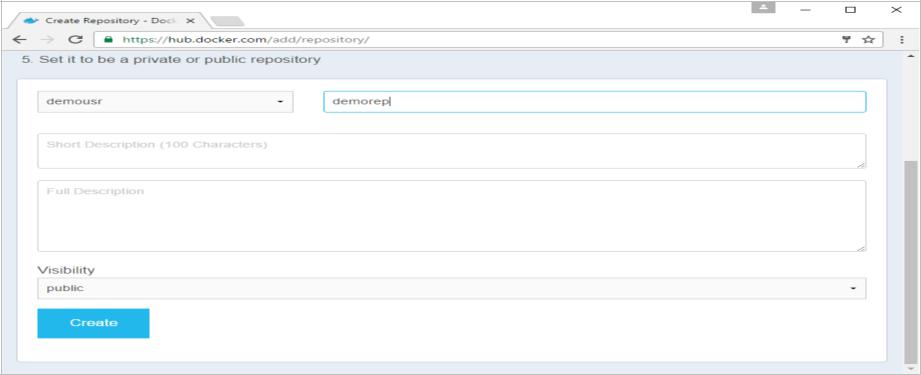
Here, we have our **myimage:0.1** image which was created as a part of the “Building Docker Files” chapter. Let’s use this to upload to the Docker public repository.

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

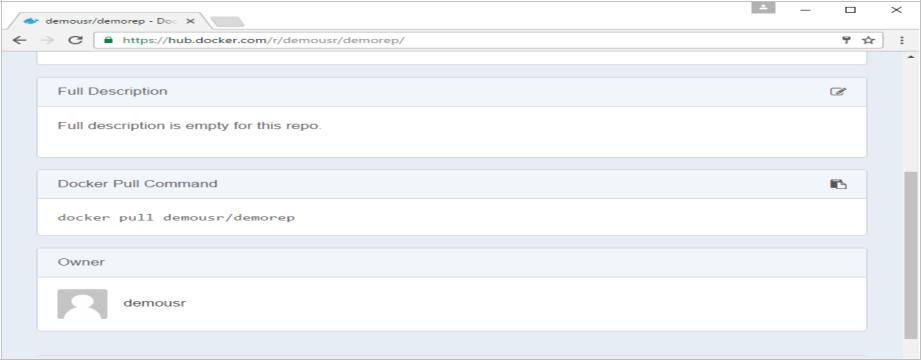
**Step 1**: Log into Docker Hub and create your repository. This is the repository where yourimage will be stored. Go to <https://hub.docker.com/>and log in with your credentials.



**Step 2**: Click the button "Create Repository" on the above screen and create a repositorywith 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.



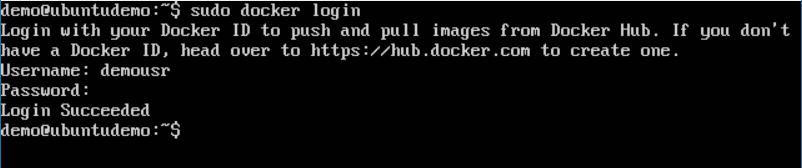
The **pull** command which will be used in our repository is as follows:

docker pull demousr/demorep

**Step 3**: Now go back to the Docker Host. Here we need to tag our **myimage** to the newrepository 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 thecommand 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 theDockerHub 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 betagged to.

**Return Value**

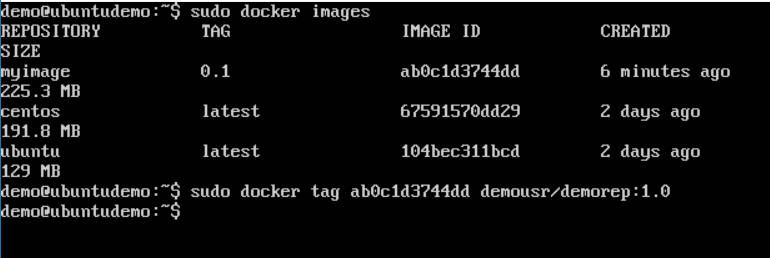
None

**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 theDocker Hub.

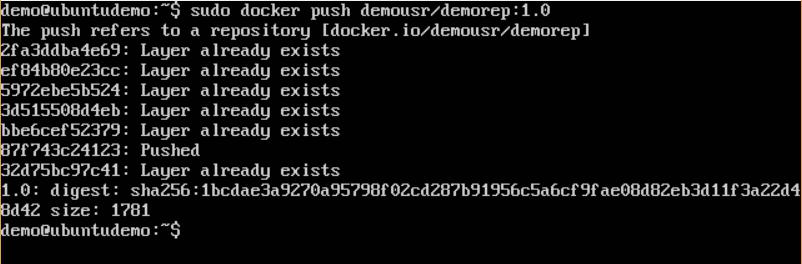
**Return Value**

The long ID of the repository pushed to Docker Hub.

**Example**

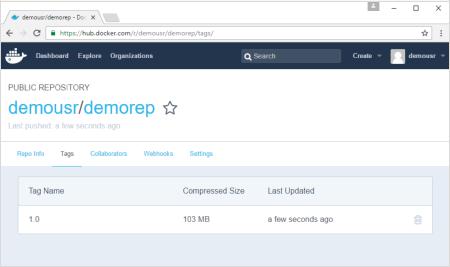
sudo docker push demousr/demorep:1.0

**Output**



59

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.



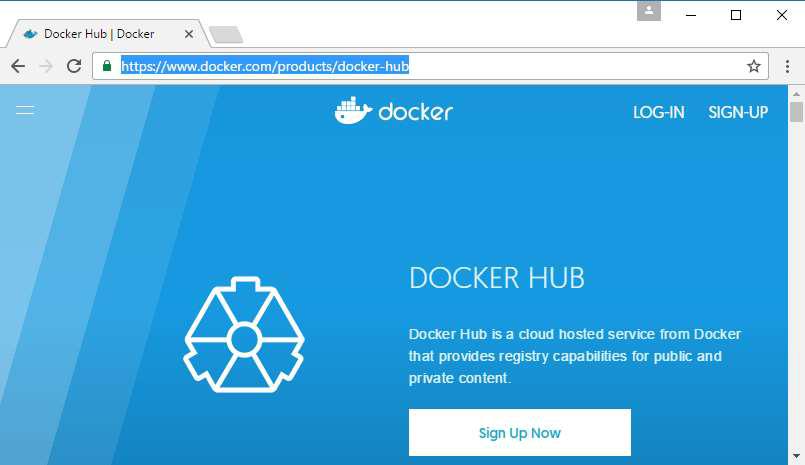
From the above screenshot, you can see that the Docker **pull** command has taken our new repository from the Docker Hub and placed it on our machine.

**Docker – Managing Ports**

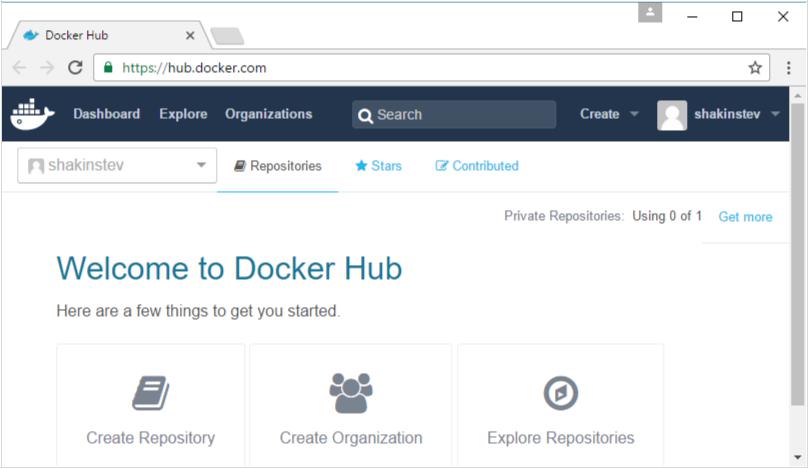
In Docker, the containers themselves can have applications running on ports. When you run a container, if you want to access the application in the container via a port number, you need to map the port number of the container to the port number of the Docker host. Let’s look at an example of how this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

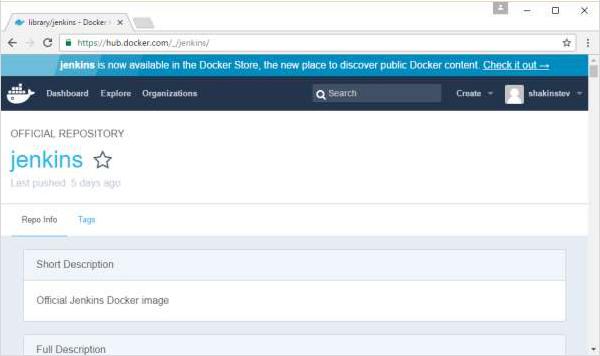
**Step 1**: First, you need to do a simple sign-up on Docker Hub.



**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.

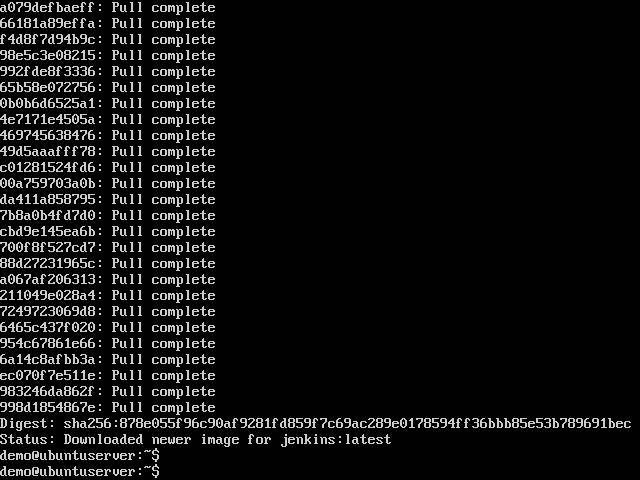


**Step 4**: If you scroll down on the same page, you can see the Docker **pull** command. Thiswill be used to download the Jenkins Image onto the local Ubuntu server.



**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 theDocker **inspect command** to inspect the image.

Let’s now learn more about this **inspect** command.

**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**

sudo 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.

