**Docker**

* 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.
* Container allows a developer to package up an application with all of the parts it needs such as libraries and other dependencies and ship it all out as one package.
* Docker makes the deployment of the s/w very easy , efficient and reliable.
* 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 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.

**Installation of Docker:**

**On a Linux machine:**

Step1:

Before installing Docker, you first have to ensure that you have the right Linux kernel version running. Docker is only designed to run on Linux kernel version 3.8 and higher. We can do this by running the following command.

uname -a

Step2:

You need to update the OS with the latest packages, which can be done via the following command

sudo apt-get update

Step3:

The next step is to install the necessary certificates that will be required to work with the Docker site later on to download the necessary Docker packages. It can be done with the following command.

sudo apt-get install apt-transport-https ca-certificates

Step4:

The next step is to add the new GPG key. This key is required to ensure that all data is encrypted when downloading the necessary packages for Docker.

The following command will download the key with the ID 58118E89F3A912897C070ADBF76221572C52609D from the **keyserver** hkp://ha.pool.sks-keyservers.net:80 and adds it to the **adv** keychain. Please note that this particular key is required to download the necessary Docker packages.

Sudo apt-key adv \ --keyserver hpk://ha.pool.sks-keyservers.net:80 \ -recv-keys 58118E89F3A912897C070ADBF76221572C52609D

Step5:

Next, depending on the version of Ubuntu you have, you will need to add the relevant site to the **docker.list** for the **apt package manager**, so that it will be able to detect the Docker packages from the Docker site and download them accordingly.

* Precise 12.04 (LTS)- deb [https://apt.dockerproject.org/repo](https://apt.dockerproject.org/repo/) ubuntu-precise main
* Trusty 14.04 (LTS) ─ deb <https://apt.dockerproject.org/repo/> ubuntu-trusty main
* Wily 15.10 ─ deb [https://apt.dockerproject.org/repo](https://apt.dockerproject.org/repo/) ubuntu-wily main
* Xenial 16.04 (LTS) - [https://apt.dockerproject.org/repo](https://apt.dockerproject.org/repo/) ubuntu-xenial main

According to to my OS version no I’ll select the url from where I want to download the docker packages.

And the command is:

echo "deb https://apt.dockerproject.org/repo ubuntu-trusty main”

| sudo tee /etc/apt/sources.list.d/docker.list

Step6:

Next, we issue the **apt-get update command** to update the packages on the Ubuntu system.

Step7:

If you want to verify that the package manager is pointing to the right repository, you can do it by issuing the **apt-cache command**.

**apt-cache policy docker-engine**

Step8:

Issue the **apt-get update command** to ensure all the packages on the local system are up to date.

Step9:

For Ubuntu Trusty, Wily, and Xenial, we have to install the linux-image-extra-\* kernel packages, which allows one to use the **aufs storage driver**. This driver is used by the newer versions of Docker.

It can be done by using the following command.

sudo apt-get install linux-image-extra-$(uname -r)

linux-image-extra-virtual

Step10:

The final step is to install Docker and we can do this with the following command.

sudo apt-get install –y docker-engine

Here, **apt-get** uses the install option to download the Docker-engine image from the Docker website and get Docker installed.

The Docker-engine is the official package from the Docker Corporation for Ubuntu-based systems.

**Some key Commands:**

1. Check docker version

sudo docker version

1. Docker Info: To see more information on the Docker running on the system

sudo docker info

Return values(Number of containers, Number of images, The storage driver used by Docker, The root directory used by Docker, The execution driver used by Docker)

**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.

The official site for Docker hub is:

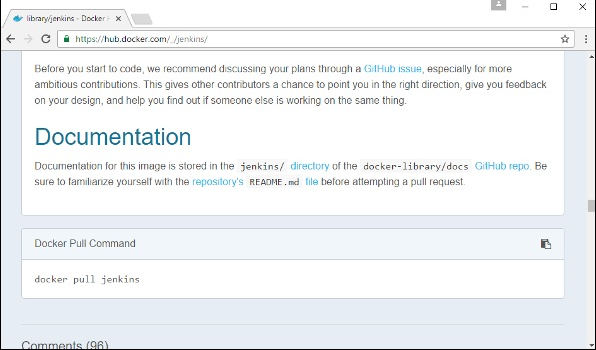
<https://www.docker.com/community-edition#/add_ons>

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

**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. This will be used to download the Jenkins image onto the local Ubuntu server.



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

sudo docker pull Jenkins

Here, **jenkins** is the name of the image we want to download from Docker hub and install on our Ubuntu machine.

To run Jenkins, you need to run the following command –

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

**-p** is used to map the port number of the internal Docker image to our main Ubuntu server so that we can access the container accordingly.

**DOCKER IMAGE:**

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.

**Example-1:**

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.

**Example-2:**

Now let’s look at how we can use the CentOS image available in Docker Hub to run CentOS on our Ubuntu machine

sudo docker run centos –it /bin/bash

* 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.

**Some key Commands:**

* Displaying docker images:

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

Syntax: docker images

Example: sudo docker images

Output:



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:

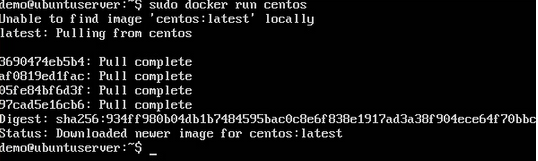
Images can be downloaded from Docker Hub using the Docker **run** command. Let’s see in detail how we can do this.

Syntax: docker run image

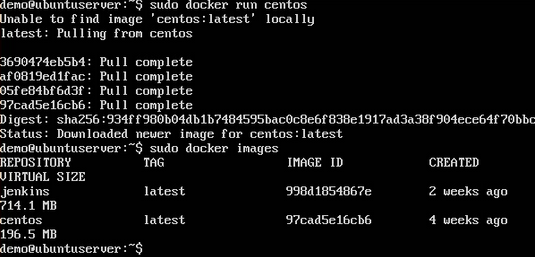
**Image** − This is the name of the image which is used to run the container.

Example: sudo docker run centos

Output



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

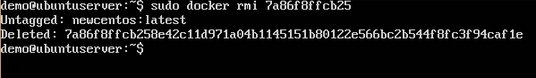
The Docker images on the system can be removed via the **docker rmi** command. Let’s look at this command in more detail.

Syntax: docker rmi<image\_id>

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

Example: sudo docker rmi 7a86f8ffcb25

Output:



* Docker images -q

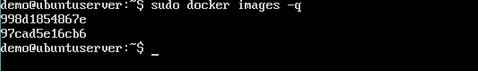
This command is used to return only the Image ID’s of the images.

Syntax: docker images –q

Example: docker images –q

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

Output:



* Docker Inspect:

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

Syntax: docker inspect **Repository**

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

Example: sudo docker inspect **jenkins**

Output:



**Docker container:**

* Containers are instances of Docker images that can be run using the Docker run command.
* The basic purpose of Docker is to run 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

**Note:**

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

**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

Example: sudo docker ps

**docker ps –a:**

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

Syntax: sudo docker ps -a

### Output



**Docker history:**

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

Syntax: docker history imageid

Example: sudo docker history centos

Output:



**docker top:**

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

Syntax: docker top **ContainerID**

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

Example: sudo docker top 9f215ed0b0d3

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



**docker stop:**

This command is used to stop a running container.

Syntax: docker stop ContainerID

Example: sudo docker stop 9f215ed0b0d3

Output: The above command will stop the Docker container **9f215ed0b0d3**



**Docker rm:**

This command is used to delete a container.

Syntax: docker rm ContainerID

Example: sudo docker rm 9f215ed0b0d3

Output: The output will give the ID of the removed container.



**Docker stats:**

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

Syntax: docker stats ContainerID

Example: sudo docker stats 9f215ed0b0d3

Output: The output will show the CPU and Memory utilization of the Container



**docker attach:**

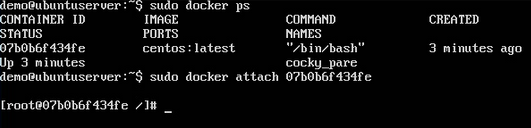
This command is used to attach to a running container.

Syntax: docker attach ContainerID

Example: sudo docker attach 07b0b6f434fe

**Output:**

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



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

Example: sudo docker pause 07b0b6f434fe

Output:



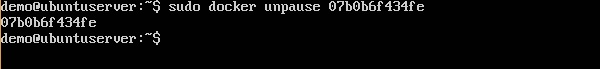
**docker unpause:**

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

Syntax: docker unpause ContainerID

Example: sudo docker unpause 07b0b6f434fe

Output:



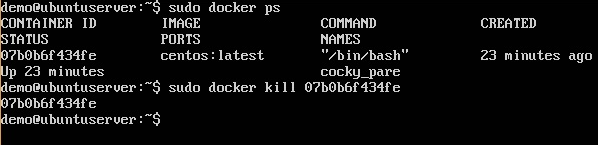
**docker kill:**

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

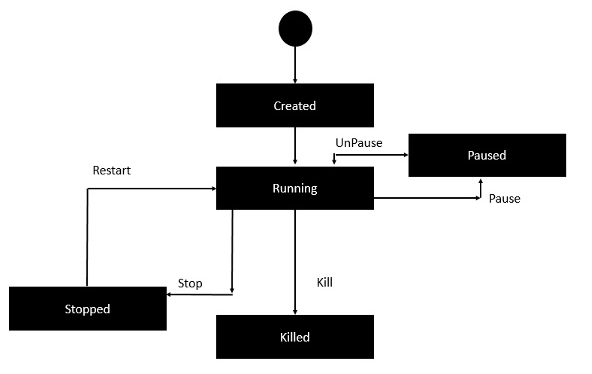
Syntax: docker kill ContainerID

Example: sudo docker kill 07b0b6f434fe

Output:



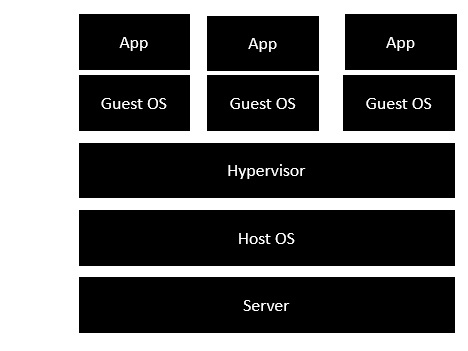
**Docker**- **container Lifecycle:**



* 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.

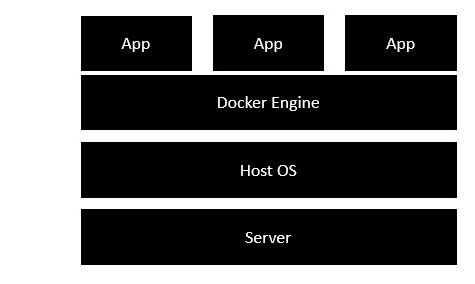
**Docker Architecture:**

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.

The good thing about the Docker engine is that it is designed to work on various operating systems. We have already seen the installation on Windows and seen all the Docker commands on Linux systems. Now let’s see the various Docker commands on the Windows OS.

**MISC:**

## Run docker Images

## docker run it ubuntu /bin/bash

## Listing all containers:

## docker ps

## Stopping a container

## Docker stop <Container\_id>

## Docker Configuration:

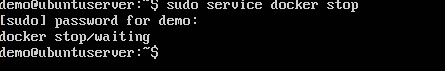
**service docker stop:**

This command is used to stop the Docker daemon process.

**Syntax:** service docker stop

**Example:** Sudo service docker stop

**Output:** A message showing that the Docker process has stopped.



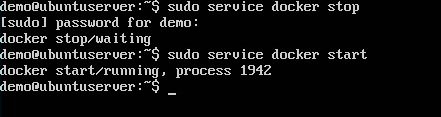
**service docker start:**

This command is used to start docker daemon process.

**Syntax:** service docker start

**Example:** sudo service docker start

**Output:** A message showing that the Docker process has started.



## By default, when you launch a container, you will also use a shell command while launching the container as shown below.

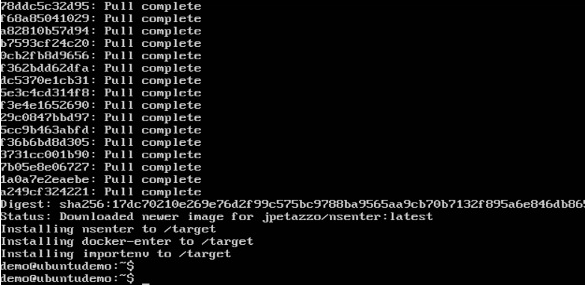
## 

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

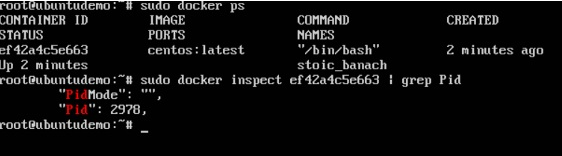
sudo docker run –it centos /bin/bash

* 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

* + **-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.

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

## Output:

## 

* 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:**

* 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.

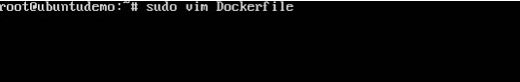


* But Docker also gives you the capability to create your own Docker images, and it can be done with the help of Docker Files. A Docker File is a simple text file with instructions on how to build your images.

Please follow below steps to create 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.



**Step2:**

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 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.

**Step3:**

Save the file.

And your docker image is created.