What is Docker?

Docker is an **open-source centralized platform designed** to create, deploy, and run applications. Docker uses **container** on the host's operating system to run applications. It allows applications to use the same **Linux kernel** as a system on the host computer, rather than creating a whole virtual operating system. Containers ensure that our application works in any environment like development, test, or production.

25.9M

598

Features of Java - Javatpoint

Docker includes components such as **Docker client, Docker server, Docker machine, Docker hub, Docker composes,** etc.

Let's understand the Docker containers and virtual machine.

Docker Containers

Docker containers are the **lightweight** alternatives of the virtual machine. It allows developers to package up the application with all its libraries and dependencies, and ship it as a single package. The advantage of using a docker container is that you don't need to allocate any RAM and disk space for the applications. It automatically generates storage and space according to the application requirement.

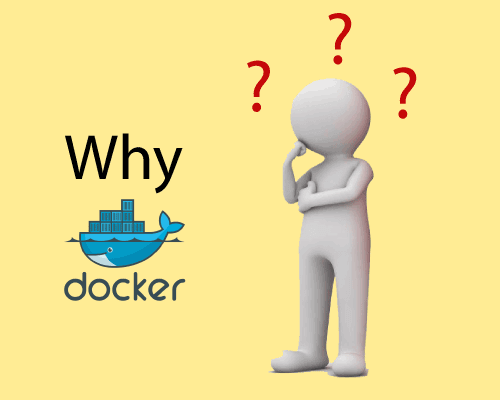
Virtual Machine

A virtual machine is a software that allows us to install and use other operating systems (Windows, Linux, and Debian) simultaneously on our machine. The operating system in which virtual machine runs are called virtualized operating systems. These virtualized operating systems can run programs and preforms tasks that we perform in a real operating system.

Containers Vs. Virtual Machine

|  |  |
| --- | --- |
| **Containers** | **Virtual Machine** |
| Integration in a container is faster and cheap. | Integration in virtual is slow and costly. |
| No wastage of memory. | Wastage of memory. |
| It uses the same kernel, but different distribution. | It uses multiple independent operating systems. |

Why Docker?



Docker is designed to benefit both the Developer and System Administrator. There are the following reasons to use Docker -

* Docker allows us to easily install and run software without worrying about setup or dependencies.
* Developers use Docker to eliminate machine problems, i.e. "**but code is worked on my laptop**." when working on code together with co-workers.
* Operators use Docker to run and manage apps in isolated containers for better compute density.
* Enterprises use Docker to securely built agile software delivery pipelines to ship new application features faster and more securely.
* Since docker is not only used for the deployment, but it is also a great platform for development, that's why we can efficiently increase our customer's satisfaction.

Advantages of Docker

There are the following advantages of Docker -

* It runs the container in seconds instead of minutes.
* It uses less memory.
* It provides lightweight virtualization.
* It does not a require full operating system to run applications.
* It uses application dependencies to reduce the risk.
* Docker allows you to use a remote repository to share your container with others.
* It provides continuous deployment and testing environment.

Disadvantages of Docker

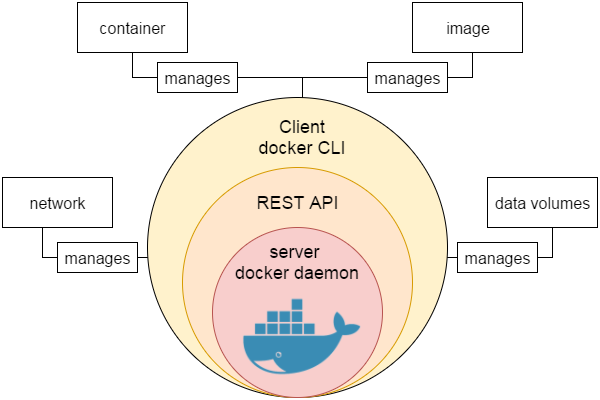
There are the following disadvantages of Docker -

* It increases complexity due to an additional layer.
* In Docker, it is difficult to manage large amount of containers.
* Some features such as container self -registration, containers self-inspects, copying files form host to the container, and more are missing in the Docker.
* Docker is not a good solution for applications that require rich graphical interface.
* Docker provides cross-platform compatibility means if an application is designed to run in a Docker container on Windows, then it can't run on Linux or vice versa.

Docker Engine

It is a client server application that contains the following major components.

* A server which is a type of long-running program called a daemon process.
* The REST API is used to specify interfaces that programs can use to talk to the daemon and instruct it what to do.
* A command line interface client.



Prerequisite

Before learning Docker, you must have the fundamental knowledge of Linux and programming languages such as java, php, python, ruby, etc.

# Docker Architecture

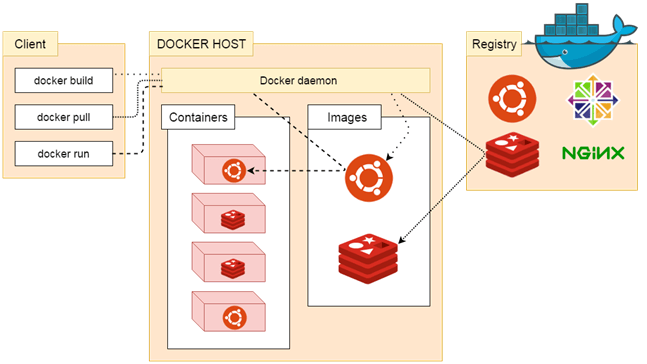
Before learning the Docker architecture, first, you should know about the Docker Daemon.

### What is Docker daemon?

Docker daemon runs on the host operating system. It is responsible for running containers to manage docker services. Docker daemon communicates with other daemons. It offers various Docker objects such as images, containers, networking, and storage. s

### Docker architecture

Docker follows Client-Server architecture, which includes the three main components that are **Docker Client**, **Docker Host**, and **Docker Registry**.



### 1. Docker Client

Docker client uses **commands** and **REST APIs** to communicate with the Docker Daemon (Server). When a client runs any docker command on the docker client terminal, the client terminal sends these docker commands to the Docker daemon. Docker daemon receives these commands from the docker client in the form of command and REST API's request.

10 Sec

OOPs Concepts in Java

#### Note: Docker Client has an ability to communicate with more than one docker daemon.

Docker Client uses Command Line Interface (CLI) to run the following commands -

docker build

docker pull

docker run

### 2. Docker Host

Docker Host is used to provide an environment to execute and run applications. It contains the docker daemon, images, containers, networks, and storage.

### 3. Docker Registry

Docker Registry manages and stores the Docker images.

There are two types of registries in the Docker -

**Pubic Registry -** Public Registry is also called as **Docker hub**.

**Private Registry -** It is used to share images within the enterprise.

## Docker Objects

There are the following Docker Objects -

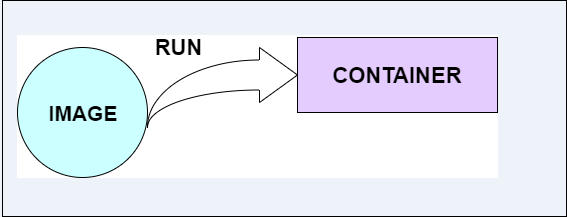
### Docker Images

Docker images are the **read-only binary templates** used to create Docker Containers. It uses a private container registry to share container images within the enterprise and also uses public container registry to share container images within the whole world. Metadata is also used by docket images to describe the container's abilities.

### Docker Containers

Containers are the structural units of Docker, which is used to hold the entire package that is needed to run the application. The advantage of containers is that it requires very less resources.

In other words, we can say that the image is a template, and the container is a copy of that template.



### Docker Networking

Using Docker Networking, an isolated package can be communicated. Docker contains the following network drivers -

* **Bridge -** Bridge is a default network driver for the container. It is used when multiple docker communicates with the same docker host.
* **Host -** It is used when we don't need for network isolation between the container and the host.
* **None -** It disables all the networking.
* **Overlay -** Overlay offers Swarm services to communicate with each other. It enables containers to run on the different docker host.
* **Macvlan -** Macvlan is used when we want to assign MAC addresses to the containers.

### Docker Storage

Docker Storage is used to store data on the container. Docker offers the following options for the Storage -

* **Data Volume -** Data Volume provides the ability to create persistence storage. It also allows us to name volumes, list volumes, and containers associates with the volumes.
* **Directory Mounts -** It is one of the best options for docker storage. It mounts a host's directory into a container.
* **Storage Plugins -** It provides an ability to connect to external storage platforms.

# How to install docker on Windows

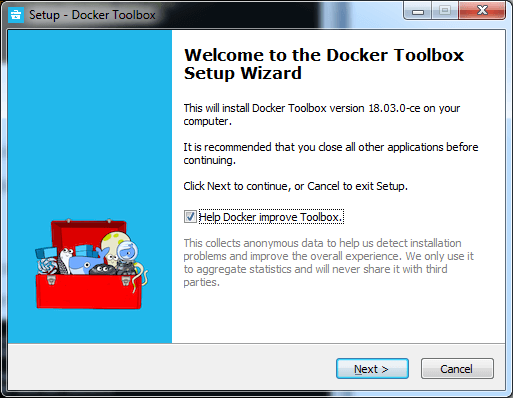
We can install docker on any operating system like **Windows, Linux,** or **Mac**. Here, we are going to install docker-engine on **Windows**. The main advantage of using Docker on Windows is that it provides an ability to run natively on Windows without any kind of virtualization. To install docker on windows, we need to download and install the **Docker Toolbox**.

Follow the below steps to install docker on windows -

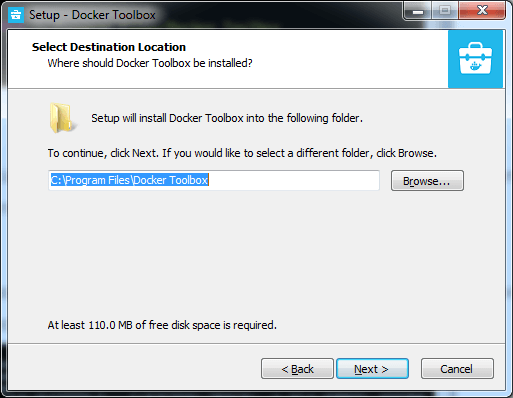
**Step 1:** Click on the below link to download DockerToolbox.exe. <https://download.docker.com/win/stable/DockerToolbox.exe>

**Step 2:** Once the **DockerToolbox.exe** file is downloaded, **double click** on that file. The following window appears on the screen, in which click on the **Next**.

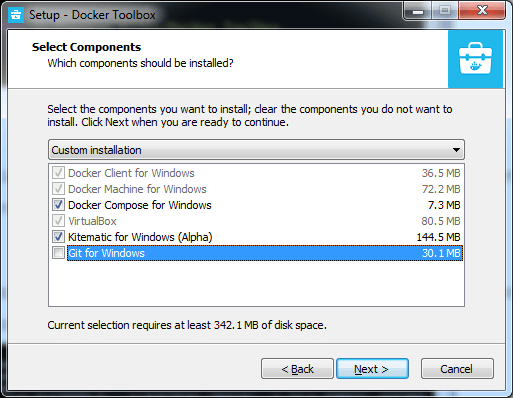
Hello Java Program for Beginners



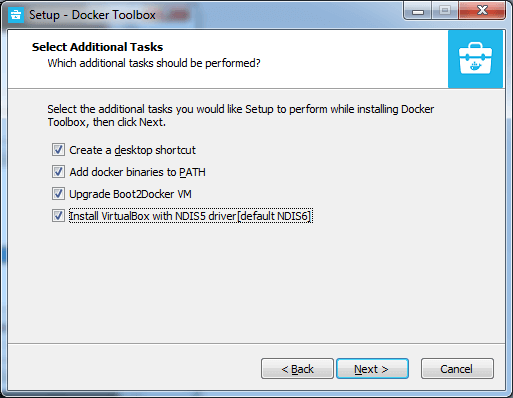
**Step 3: Browse the location** where you want to install the Docker Toolbox and click on the Next.



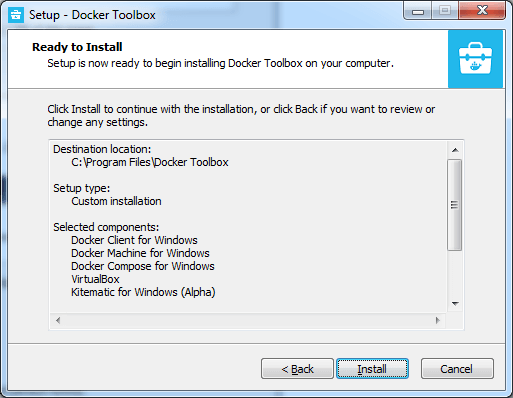
**Step 4: Select the components** according to your requirement and click on the **Next**.



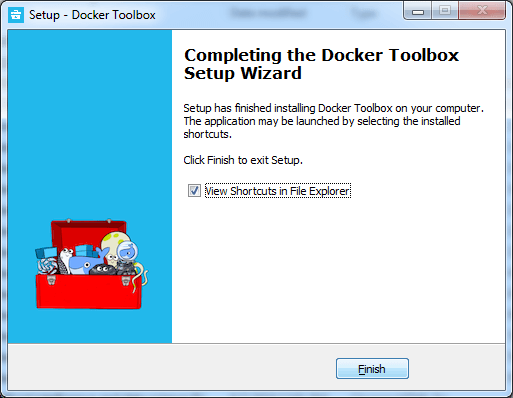
**Step 5: Select Additional Tasks** and click on the **Next**.



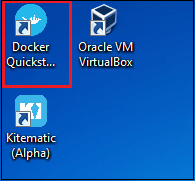
**Step 6:** The Docker Toolbox is ready to install. Click on **Install**.



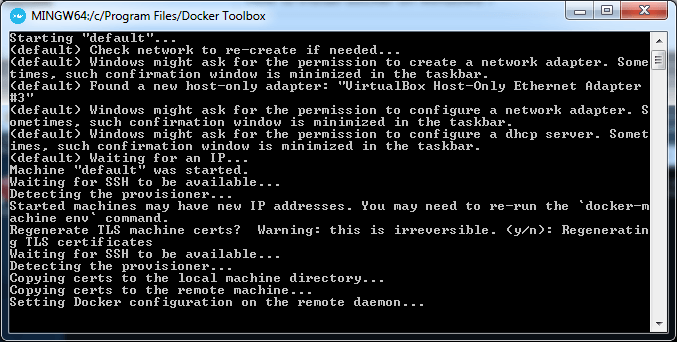
**Step 7:** Once the installation is completed, the following Wizard appears on the screen, in which click on the **Finish**.



**Step 8:** After the successful installation, three icons will appear on the screen that are: **Docker Quickstart Terminal, Kitematic (Alpha),** and **OracleVM VirtualBox**. **Double click** on the Docker Quickstart Terminal.



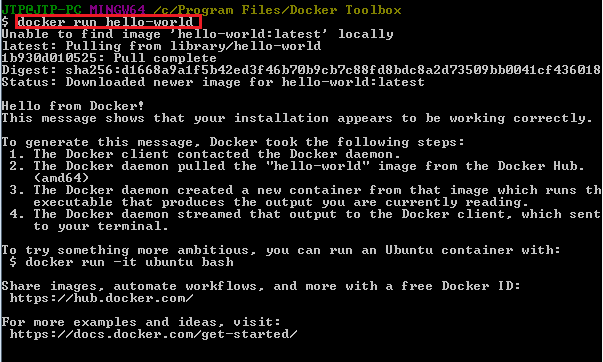
**Step 9:** A Docker Quickstart Terminal window appears on the screen.



To verify that the docker is successfully installed, type the below command and press enter key.

1. docker run hello-world

The following output will be visible on the screen, otherwise not.



You can check the Docker version using the following command.

1. docker -version

install docker on Windows

# Docker Container and Image

Docker container is a running instance of an image. You can use Command Line Interface (CLI) commands to run, start, stop, move, or delete a container. You can also provide configuration for the network and environment variables. Docker container is an isolated and secure application platform, but it can share and access to resources running in a different host or container.

An image is a read-only template with instructions for creating a Docker container. A docker image is described in text file called a **Dockerfile**, which has a simple, well-defined syntax. An image does not have states and never changes. Docker Engine provides the core Docker technology that enables images and containers.

You can understand container and image with the help of the following command.

1. $ docker run hello-world

The above command **docker run hello-world** has three parts.

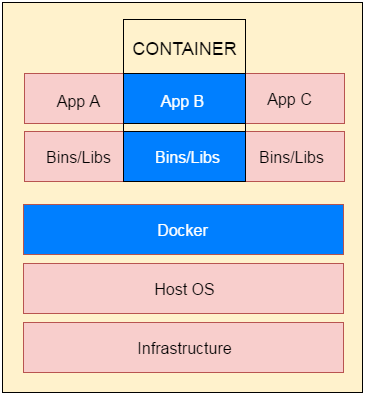
OOPs Concepts in Java

1) **docker:** It is docker engine and used to run docker program. It tells to the operating system that you are running docker program.

2) **run:** This subcommand is used to create and run a docker container.

3) **hello-world:** It is a name of an image. You need to specify the name of an image which is to load into the container.

### Docker Container



**Fig**: docker-container

# Docker Dockerfile

A Dockerfile is a text document that contains commands that are used to assemble an image. We can use any command that call on the command line. Docker builds images automatically by reading the instructions from the Dockerfile.

The docker build command is used to build an image from the Dockerfile. You can use the -f flag with docker build to point to a Dockerfile anywhere in your file system.

1. $ docker build -f /path/to/a/Dockerfile .

## Dockerfile Instructions

The instructions are not case-sensitive but you must follow conventions which recommend to use uppercase.

Docker runs instructions of Dockerfile in top to bottom order. The first instruction must be **FROM** in order to specify the Base Image.

SQL CREATE TABLE

A statement begin with # treated as a comment. You can use RUN, CMD, FROM, EXPOSE, ENV etc instructions in your Dockerfile.

Here, we are listing some commonly used instructions.

### FROM

This instruction is used to set the Base Image for the subsequent instructions. A valid Dockerfile must have FROM as its first instruction.

Ex.

1. FROM ubuntu

### LABEL

We can add labels to an image to organize images of our project. We need to use LABEL instruction to set label for the image.

Ex.

1. LABEL vendorl = "JavaTpoint"

### RUN

This instruction is used to execute any command of the current image.

Ex.

1. RUN /bin/bash -c 'source $HOME/.bashrc; echo $HOME'

### CMD

This is used to execute application by the image. We should use CMD always in the following form

1. CMD ["executable", "param1", "param2"?]

This is preferred way to use CMD. There can be only one CMD in a Dockerfile. If we use more than one CMD, only last one will execute.

### COPY

This instruction is used to copy new files or directories from source to the filesystem of the container at the destination.

Ex.

1. COPY abc/ /xyz

**Rules**

* The source path must be inside the context of the build. We cannot COPY ../something /something because the first step of a docker build is to send the context directory (and subdirectories) to the docker daemon.
* If source is a directory, the entire contents of the directory are copied including filesystem metadata.

### WORKDIR

The WORKDIR is used to set the working directory for any RUN, CMD and COPY instruction that follows it in the Dockerfile. If work directory does not exist, it will be created by default.

We can use WORKDIR multiple times in a Dockerfile.

Ex.

1. WORKDIR /var/www/html

# Docker Java Application Example

As, we have mentioned earlier that docker can execute any application.

Here, we are creating a Java application and running by using the docker. This example includes the following steps.

1. **Create a directory**

Directory is required to organize files. Create a director by using the following command.

* 1. $ mkdir  java-docker-app

See, screen shot for the above command.

Docker Java application 1

1. **Create a Java File**

Now create a Java file. Save this file as **Hello.java** file.

**// Hello.java**

* 1. **class** Hello{
  2. **public** **static** **void** main(String[] args){
  3. System.out.println("This is java app \n by using Docker");
  4. }
  5. }

Save it inside the directory **java-docker-app** as Hello.java.

1. **Create a Dockerfile**

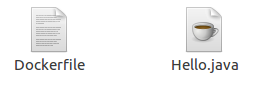
After creating a Java file, we need to create a Dockerfile which contains instructions for the Docker. Dockerfile does not contain any file extension. So, save it simple with **Dockerfile** name.

**// Dockerfile**

* 1. FROM java:8
  2. COPY . /var/www/java
  3. WORKDIR /var/www/java
  4. RUN javac Hello.java
  5. CMD ["java", "Hello"]

Write all instructions in uppercase because it is convention. Put this file inside **java-docker-app** directory. Now we have Dockerfile parallel to Hello.java inside the **java-docker-app** directory.

See, your folder inside must look like the below.

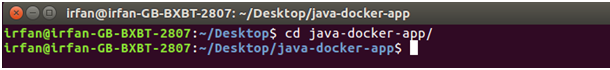


1. **Build Docker Image**

After creating Dockerfile, we are changing working directory.

* 1. $ cd   java-docker-app

See, the screen shot.



Now, create an image by following the below command. we must login as root in order to create an image. In this example, we have switched to as a root user. In the following command, **java-app**is name of the image. We can have any name for our docker image.

* 1. $ docker build -t java-app .

See, the screen shot of the above command.



After successfully building the image. Now, we can run our docker image.

1. **Run Docker Image**

After creating image successfully. Now we can run docker by using run command. The following command is used to run java-app.

* 1. $ docker run java-app

See, the screen shot of the above command.



Here, we can see that after running the java-app it produced an output.

Now, we have run docker image successfully on your system. Apart from all these you can also use other commands as well.

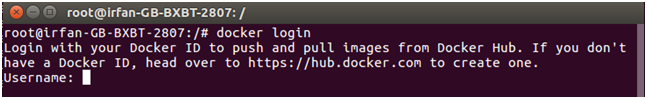
# Docker Push Repository

We can push our Docker image to global repository. It is a public repository provided by Docker officially. It allows us to put our docker image on the server. It is helpful when we want to access our docker image from global. Follow the following steps to push custom image on the Docker hub.

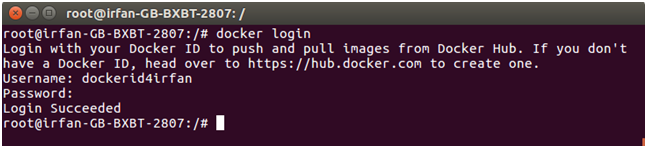
1. **login to hub.docker.com**

We need to login to our account of Docker hub. If you don't have, **create it first.**

* 1. $ docker login



It will ask for username. Enter the dockerid here and press enter.



After providing username, it asks for password. Enter your account password here and it will show you your login status as succeeded.

1. **Tag Docker Image**

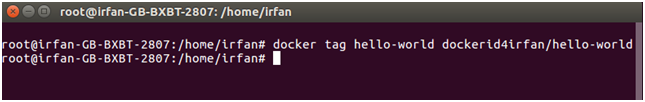
After login, we need to tag our docker image that we want to push. Following command is used to tag the docker image.

* 1. $ docker tag image-name username/image-name

**username** refers to our dockerid or the username which is used to login.

**image-name** is the name of our docker image present on our system.

See, screen shot of the above command.



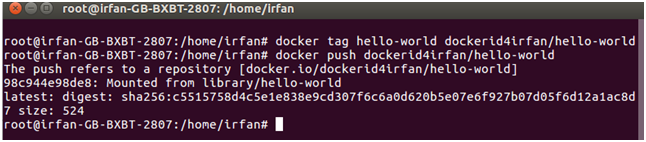
In the above command, we have tagged docker image **hello-world**. Now, we need to push it to the repository. Let?s see it in the below command.

1. **Push Docker Image**

The following command is used to push docker image to docker hub repository.

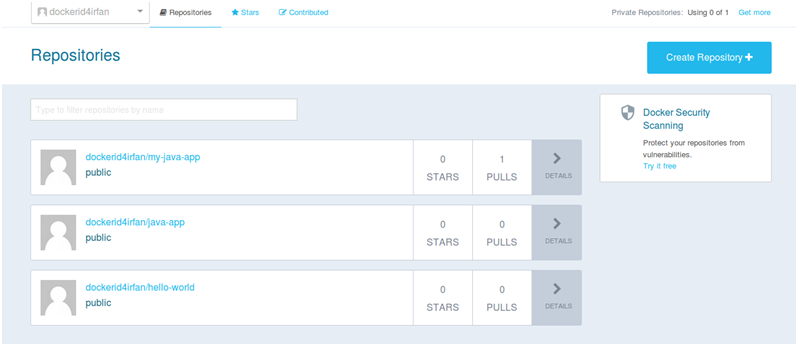
* 1. $ docker push  username/image-name

See, screen shot of the above command.



In the above screen shot, we can see that docker image has been pushed successfully.

Now, login into our account at hub.docker.com and check our dashboard. It will have a new docker image named **dockerid4irfan/hello-world.**



Look at the screen shot, it has the newest one docker image which is just pushed. On the top, the first one is the newest image.

# Docker Useful Commands

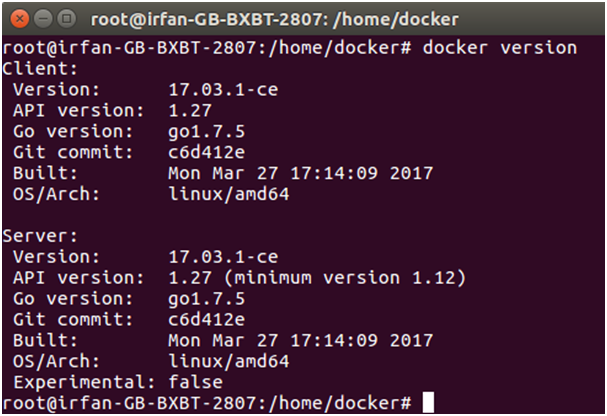
Docker is natively Linux based software so that it provides commands to interact and work in the client-server environment.

Here, we have listed some important and useful Docker commands.

### Check Docker version

1. $ docker version

It shows docker version for both client and server. As given in the following image.



### Build Docker Image from a Dockerfile

1. $ docker build -t image-name docker-file-location

**-t**: it is used to tag Docker image with the provided name.

OOPs Concepts in Java

### Run Docker Image

1. $ docker run -d image-name

**-d** : It is used to create a daemon process.

### Check available Docker images

1. $ docker images

### Check for latest running container

1. $ docker ps -l

**-l** : it is used to show latest available container.

### Check all running containers

1. $ docker ps -a

**-a** : It is used to show all available containers.

### Stop running container

1. $ docker stop container\_id

**container\_id** : It is an Id assigned by the Docker to the container.

### Delete an image

1. $ docker rmi image-name

### Delete all images

1. $ docker rmi $(docker images -q)

### Delete all images forcefully

1. $ docker rmi -r $(docker images -q)

**-r** : It is used to delete image forcefully.

### Delete all containers

1. $ docker rm $(docker ps -a -q)

### Enter into Docker container

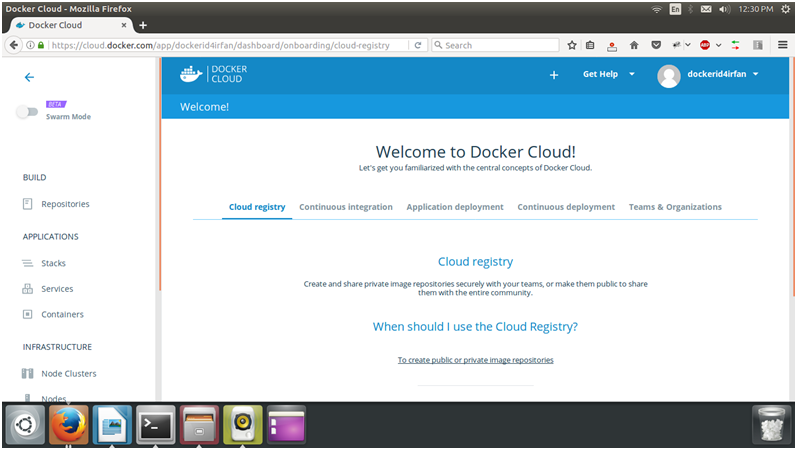
1. $ docker exec -it container-id bash

# Docker Cloud

Docker provides us the facility to store and fetch docker images on the cloud registry. We can store dockerized images either privately or publicly. It is a full GUI interface that allows us to manage builds, images, swarms, nodes and apps.

We need to have Docker ID to access and control images. If we don't have, create it first.

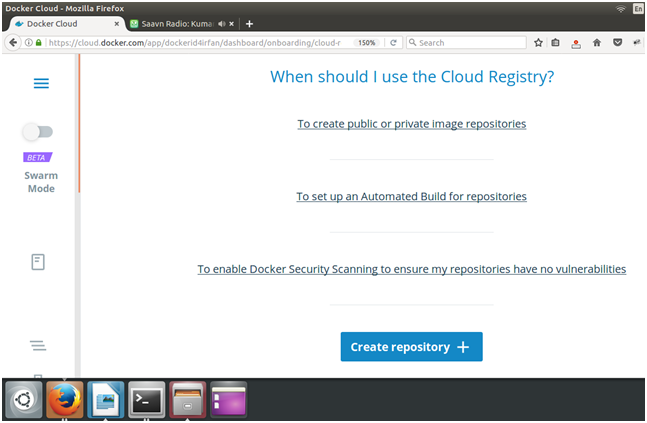
Here, in the following screenshot, we have logged in to Docker cloud. It shows a welcome page.



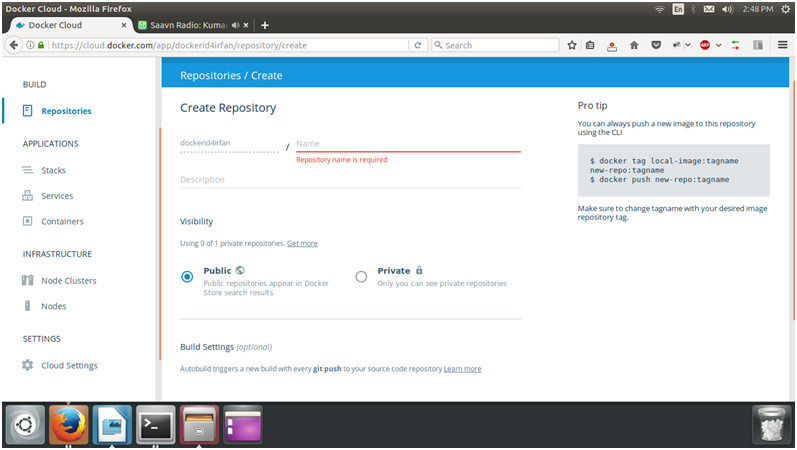
In the left panel, we can see that it provides lots of functionalities that we use on the cloud. Apart from all these, let's create a repository first.

## Creating Repository

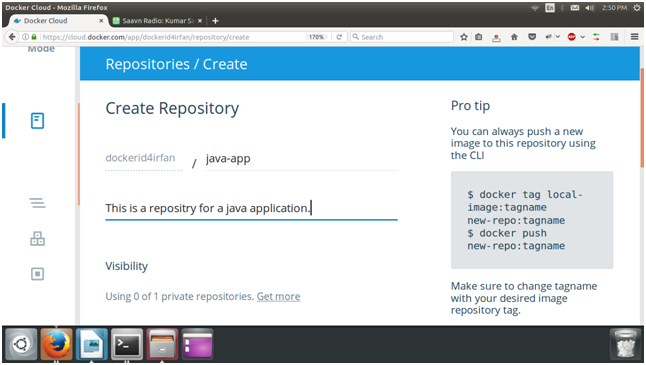
To create Docker cloud repository, click on the create repository +button available on the welcome page at the bottom.



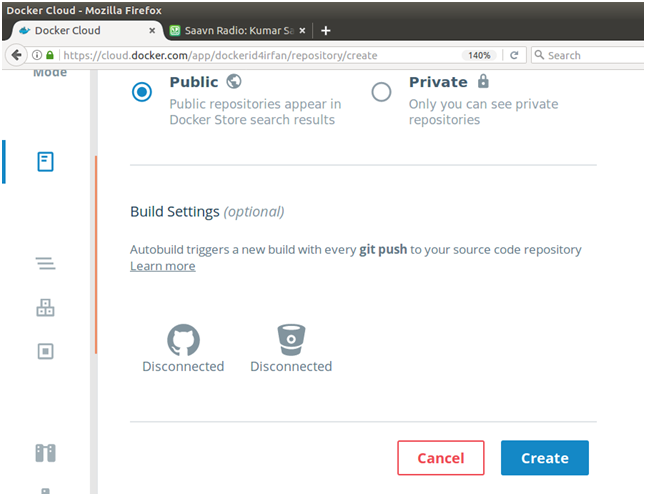
After clicking, it displays a form to enter the name of the repository. The page looks looks like the following.



It asks for the repository name to create a new one. The following screen-shot show the description.



After filling the details, we should make this repository public. Now, just click on the create button at the bottom. It will create repository for us.



So, we can see that it provides the other tools also to manage and control Docker cloud.