**What is Docker?**

Build, Ship, and Run Any App, Anywhere

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers.

Containers allow 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.

By doing so, thanks to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

Docker is a bit like a virtual machine. But unlike a virtual machine, rather than creating a whole virtual operating system, Docker allows applications to use the same Linux kernel as the system that they're running on and only requires applications be shipped with things not already running on the host computer.

This gives a significant performance boost and reduces the size of the application.

Offical Website: <https://www.docker.com/>

Official Documentation: <https://docs.docker.com/>

**Advantages of Docker?**

Docker is a tool that is designed to benefit both developers, testers and system administrators,

making it a part of many DevOps (developers + operations) toolchains.

For developers, it means that they can focus on writing code without worrying about the system that it will ultimately be running on. It also allows them to get a head start by using one of thousands of programs already designed to run in a Docker container as a part of their application.

Docker is used to deploy art scale with ease, which also saves time, as Configurations, libraries, webservers are completely built into Image itself. Even JRE as well may be included in Image itself. Reduces/avoids last minute hurdles, which inturn improves productivity.

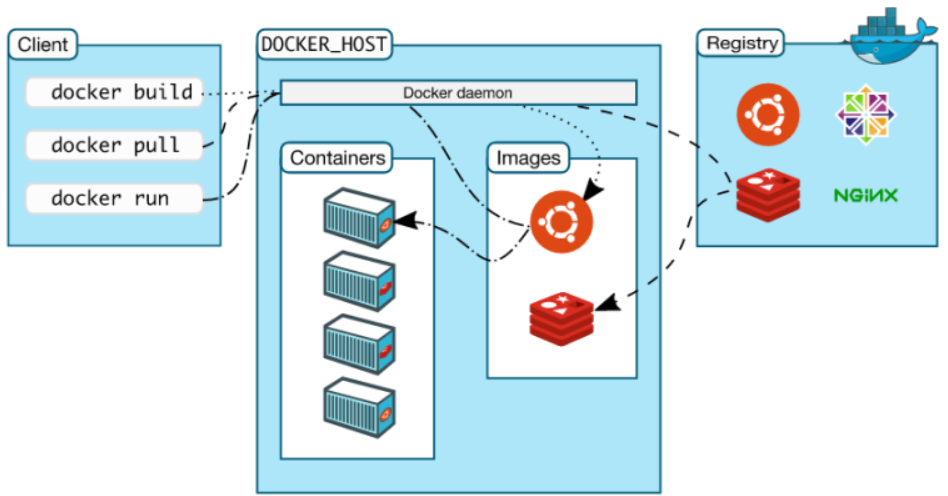
For operations staff, Docker gives flexibility and potentially reduces the number of systems needed because of its small footprint and lower overhead.

Deployment is easy as it will be a single image file, which includes, and behavior will be same irrespective of Host Operating system or evironment,

there by number of surprises or side effects due to the environment are minimal or zero.

With less effort DevOps can deploy the readymade image onto multiple Nodes, especially for Microservices this adds lot of value by saving time.

Same image will be used across Development, Testing, Production Dockers, and same behavior is experienced across, due to Docker Container.



**Platforms on which Docker is supported?**

1. Linux
2. Windows
3. ON various clouds like AWS, Google Cloud, Rackspace, Digital Ocean
4. Mac

**An Enterprise webapp, need to run on various Web Servers.**

**Docker CLI, Docker Daemon, Docker Container, Docker Engine,**

***Docker Daemon/Engine*** - The background service running on the host that manages building, running and distributing Docker containers. The daemon is the process that runs in the operating system to which clients talk to. dockerd is the persistent process that manages containers.

Docker uses different binaries for the daemon and client. To run the daemon you type dockerd.

***Docker Client*** - The command line tool that allows the user to interact with the daemon.

***Containers*** - Created from Docker images and run the actual application. We create a container using docker run which we did using the busybox image that we downloaded. A list of running containers can be seen using the docker pscommand.

**Registry Service(Docker Hub & DOcker Trusted Registry)**

**What is 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.

Website: <http://hub.docker.com>

**Docker Run Sample image which is already existing in docker hub**

docker run hello-world

**Docker installation & Setup?**

**Docker images**

Images may come from two sources:

* image repository, which is Docker Hub.
* you can create your own images. We will talk about this in more details, because this part is awesome. Docker images are layered, so you can build them layer by layer, and you can build your images starting from other images.

**How to Dockerize a Java Application:**

Below are steps for the same

1. Create jar file ( you can use maven for this)
2. Create Dockerfile which is used to create an image
3. Create image
4. Run the container with the image created above

**What is a Dockerfile and why you’d want to use one?**

Define Container using Dockerfile.

A Dockerfile is a text file that defines a Docker image. You’ll use a Dockerfile to create your own custom Docker image, in other words to define your custom environment to be used in a Docker container.

Below is sample Dockerfile

FROM java:8

WORKDIR /

ADD HelloWorld.jar HelloWorld.jar

EXPOSE 8080

CMD java -jar HelloWorld.jar

Now you are ready to create a Docker image, the result of building a Dockerfile and executing the Dockerfile's commands. It is constructed from a root operating system, installed applications, and commands executed in such a way that it can run your application. A Docker image serves as the basis for Docker containers and is the static template from which they are created.

You need to run in command line the following:

docker build -t helloworld

NOTE: Below are other possible commands which can be used in Dockerfile

|  |  |
| --- | --- |
| Command | Description |
| ADD | Copies a file from the host system onto the container |
| CMD | The command that runs when the container starts |
| ENTRYPOINT |  |
| ENV | Sets an environment variable in the new container |
| EXPOSE | Opens a port for linked containers |
| FROM | The base image to use in the build. This is mandatory and must be the first command in the file. |
| MAINTAINER | An optional value for the maintainer of the script |
| ONBUILD | A command that is triggered when the image in the Dcokerfile is used as a base for another image |
| RUN | Executes a command and save the result as a new layer |
| USER | Sets the default user within the container |
| VOLUME | Creates a shared volume that can be shared among containers or by the host machine |
| WORKDIR | Set the default working directory for the container |

Now run the above image created on the Docker Container using below command.

**Other Popular Docker commands:**

* docker run – Runs a command in a new container.
* docker start – Starts one or more stopped containers
* docker stop – Stops one or more running containers
* docker build – Builds an image form a Docker file
* docker pull – Pulls an image or a repository from a registry
* docker push – Pushes an image or a repository to a registry
* docker export – Exports a container’s filesystem as a tar archive
* docker exec – Runs a command in a run-time container
* docker search – Searches the Docker Hub for images
* docker attach – Attaches to a running container
* docker commit – Creates a new image from a container’s changes

Brief Reference: <https://docker-curriculum.com/>

---------------------------

**Step By Step: Creation of sample Image & running**

The first thing you need is to create a basic.java file, HelloWorld.java, and add these lines into it:

public class HelloWorld {

public static void main(String[] args){

System.out.println("Hello World :) ");

}

}

Save and compile it in the command line. From the directory in which you have created your HelloWorld.java, run the command javac HelloWorld.java.

Once you do this, you will get the HelloWorld.class file, which later we will build in .jar. But before that, we need to create a simple manifest.txt to make it packed right.

So now, in the same directory, create manifest.txt and place the following lines:

Manifest-Version: 1.0

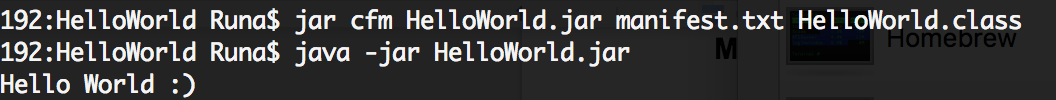
Created-By: Me

Main-Class: HelloWorld

Then, in the command line, run the following: jar cfm HelloWorld.jar manifest.txt HelloWorld.class.

And to check if everything works correctly, type java -jar HelloWorld.jar.

If everything is okay, you should see the following:



The next step is to start Docker and create a Dockerfile, a text file that contains the instructions (or commands) used to build a Docker image.

To do that, create the file with the name "Dockerfile" and place the following text in it:

FROM java:8

WORKDIR /

ADD HelloWorld.jar HelloWorld.jar

EXPOSE 8080

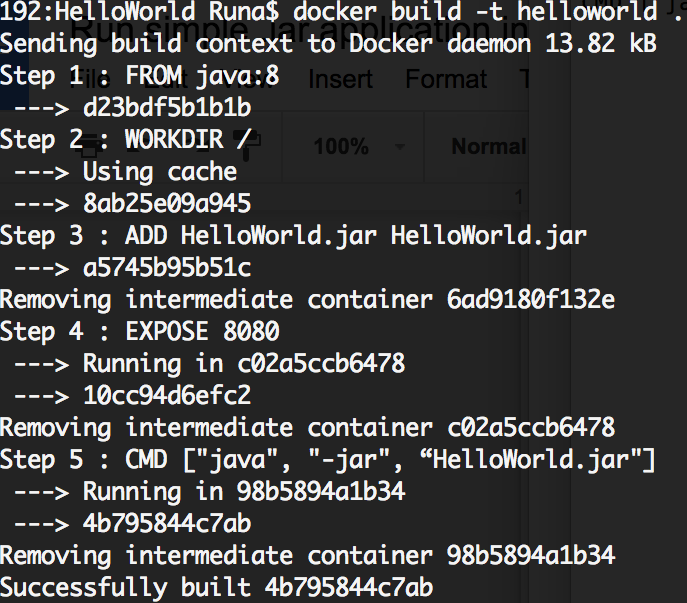
CMD java -jar HelloWorld.jar

Don’t forget to leave the empty line at the end of the file.

Now you are ready to create a Docker image, the result of building a Dockerfile and executing the Dockerfile's commands. It is constructed from a root operating system, installed applications, and commands executed in such a way that it can run your application. A Docker image serves as the basis for Docker containers and is the static template from which they are created.

You need to run in command line the following: docker build -t helloworld

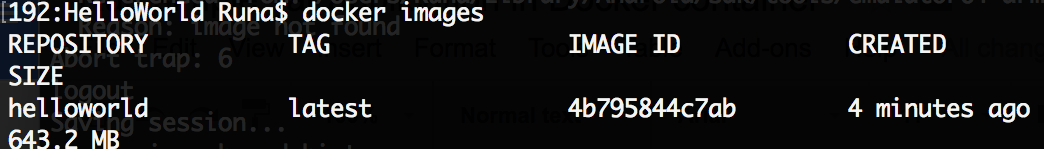
As a result, you should see this:



Then you have to create an account on [dockerhub](https://hub.docker.com/) and create the [repository](https://docs.docker.com/docker-hub/repos/) "hello-world" to push your image to your repository. Once you register and create a repository, go to command line and log in there with docker login.

Then pull that repository: docker pull /hello-world

To push your Docker image to DockerHub you need to figure out your Docker\_Image\_ID. Run the following: docker images



So you may find your image and see you Image\_Id. Now you need to tag and push your image: docker tag 4b795844c7ab /hello-world

To read more about working with it, you can go [here](https://docs.docker.com/engine/getstarted/step_six/).

Now you are ready to upload your Docker Image to DockerHub.

Just type: docker push /hello-world:latest

To check if everything works fine enter: docker run /hello-world

You must see the output: Hello World :)