**Docker Tutorial**

**Docker is a centralized platform for packaging, deploying, and running applications.**

1. Provides easy way to deploy an application

**Docker Registry, Docker Repository, Tags**

**Docker Registry/Docker Hub :** place where all repositories are there.

**Docker Repository:** place where all images/tags are present.

**Tag:** its kind of war/ear file or image.

**What is a docker?**

Docker is an open platform for developing, running and shipping applications. Docker facilitates you to separate out your applications from your infrastructure, so that you hand over it to the client instantly. In addition, using docker you can manage your infrastructure in the same ways as you manage your applications. You can also remarkably reduce the time interval between writing code and running it in production by taking the benefits of Docker’s techniques for deploying, testing and shipping code rapidly.

**What is a docker container?**

A container is a kind of software that packs up code and all its dependencies in a standard unit so the application runs from one environment to another quickly and reliably. Likewise Docker provides the ability to wrap up and run an application without affecting the reliability in an isolated environment is known as a docker container. Moreover the isolation and security assist you to run multiple containers concurrently on a given host. Containers isolate software from its original environment and ensure that it works equally regardless of differences in environments, for example between development and [staging](https://en.wikipedia.org/wiki/Deployment_environment#Staging). Containers are lightweight because they share the machine’s OS system kernel and therefore do not require an OS per application.

**What is a docker image?**

A Docker image is a lightweight, standalone, executable bundle of software that contains everything (code, runtime, system tools, system libraries and settings) which is required to run an application. Generally a container image becomes containers at runtime. Similarly, in the case of Docker containers, images become containers when they run on [Docker Engine](https://www.docker.com/products/container-runtime). In other words Docker Image is a template for the container’s specific runtime environment. Docker Images are available for both Linux and Windows-based applications, containerized application will always run the same, regardless of the infrastructure.

**What is a docker host?**

Docker host is a physical computer as a server or a virtual machine on which Docker Daemon runs. It can be a server or virtual machine in a data center or a computing device by a cloud provider or even your laptop. In fact a Docker Daemon is a component on the Docker host that is responsible for building and running containers. The daemon starts each container with the help of a container image.

**Why docker is used?**

1) Using Docker’s methodology you can reduce the time interval between development and deployment in production.  
2) Docker is an open platform for development, deployment and shipment.  
3) Docker provides the ability to wrap up and run an application without affecting the reliability in an isolated environment.  
4) With Docker, you can run multiple containers simultaneously on a given host.  
5) Docker Containers are lightweight so they increases application performance drastically.

6) Containerized application always runs consistently, regardless of the infrastructure, for instance, in both Linux and Windows-based applications.  
7) Docker Containers have their own built-in mechanisms for versioning and component reuse.  
8) Containerized application can easily be shared via the public repository as Docker Hub or private repositories.  
9) Instances of containerized apps use far less memory than virtual machines, they start and stop more quickly.  
10) Generally docker containers are absolutely suitable for the Microservices and to agile development processes.

**What is docker hub?**

[Docker Hub](https://hub.docker.com/) is a kind of repository service provided by Docker for storing, finding and sharing container images with your team. It is the world’s largest repository of container images. In addition, it assists with various content sources, including container community developers, open source projects and independent software vendors (ISV) building and distributing their code in containers. Docker hub facilitates two types of repositories : public repository and private repository. Public repositories are free to use, whereas private repositories come with subscription plans.

docker run -p 5000:5000 **in28min/hello-world-python**:0.0.1.RELEASE

**in28min/hello-world-python**:0.0.1.RELEASE : path stored in docker hub. It’s a registry where all repositories are stored

**in28min/hello-world-python : repository name.** same repository can have multiple tags

0.0.1.RELEASE : image/ tag/ release contains everything like software, libraries, dependencies might need to be able to run

**Execution of Docker command**

docker run -p 5000:5000 **in28min/hello-world-python**:0.0.1.RELEASE

when we execute above command, so docker will search for the pecific image in local initially. If it does not find that, then it will pull it from docker hub/ repo to our local. Then it will run that image. Running image is called as **container.** For same image we can have lot of container

**-p 5000:5000**

Whenever we run a container, it is part of internal Docker network, called a bridge network. So by default, all containers run inside the bridge network. You will not be able to access the container, unless the port is exposed outside. So what we are doing in here, is we're taking the container port to 5000, and mapping it to a host port.

Container port 5000, is map it to a host port - a port on the local machine – at 5000.

And the option which is used to enable us to do this is -p. -p is a shortcut for something called --**publish**.

So we're publishing a container port on to a host port.

-p **5000**:5000

**Container port : host port**

**Conclusion:**

The image is actually downloaded from Docker registry, a public Docker registry called ***Docker Hub***. A registry contains a number of repositories, and one of the repositories which is present in there is hello-world-nodejs.

So in28min/hello-world-nodejs = name of the repository

0.0.1.RELEASE = version of a specific repository.

A repository typically maps to a specific deployable unit, and you might have multiple versions of your deployable units - 0.0.1.RELEASE to 0.0.3.RELEASE.

this 0.0.1.RELEASE in here, is a tag which identifies which version of the application which version of the deployable unit.

static version of this specific image on the repository, or on your local machine is what is called **image**, and a running version of this image is called a **container**.

**Running multiple containers using multiple command prompt**

We can run multiple containers of same image at different port number like below (command prompt window1)

*In cmd window 1 2 3*

docker run -p **500 0**:5000 in28min/hello-world-python:0.0.1.RELEASE

docker run -p **5001**:5000 in28min/hello-world-python:0.0.1.RELEASE

docker run -p **5002**:5000 in28min/hello-world-python:0.0.1.RELEASE

*In cmd window 1 2 3*

docker run -p **5000**:5000 in28min/hello-world-java:0.0.1.RELEASE

docker run -p **5001**:5000 in28min/hello-world- java:0.0.1.RELEASE

docker run -p **5002**:5000 in28min/hello-world- java:0.0.1.RELEASE

we can achieve all above using single command prompt window. In above case, the containers is tied with the terminal.

We have to run all the containers in detached mode. We don’t want to tie up container with terminal

Docker run --d

**Running multiple containers using single command prompt**

Here we need to run in detached mode

**All commands**

**docker --version** : to see the version

**Following commands are to run specific appliation in seperate command prompts**

docker run -p 5000:5000 in28min/hello-world-python:0.0.1.RELEASE

docker run -p 5000:5000 in28min/hello-world-java:0.0.1.RELEASE

docker run -p 5000:5000 in28min/hello-world-nodejs:0.0.1.RELEASE

**Following command is used to run application with detached mode(detached from command prompts)**

docker run -d -p 5000:5000 in28min/hello-world-nodejs:0.0.1.RELEASE

docker run -d -p 5001:5000 in28min/hello-world-python:0.0.1.RELEASE

**command to see the specific logs**

docker logs 04e52ff9270f5810eefe1f77222852dc1461c22440d4ecd6228b5c38f09d838e

docker logs c2ba

**See all images**

docker images

**See all the containers in list format**

docker container ls

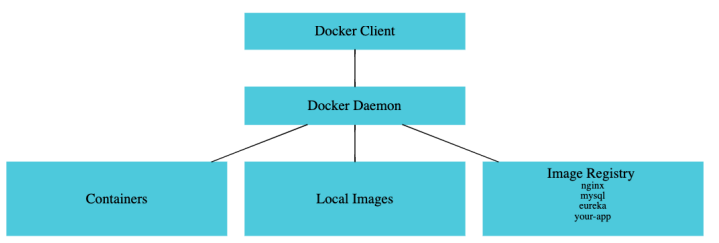
docker container ls –a

docker bydefault gives name to containers

**Stoppng contaners**

docker container stop f708b7ee1a8b

**Understanding Docker Architecture**



When send command in docker client, then it transfers it to docker daemon. Docker Daemon is responsible for doing all the necessary operation and send response back to Docker Client.

Docker provides us below advantages

1. **Standardized Application packaging**

Application packaging is same for all types of application. If we are having different microservices created in different language, the way to package is same in case of docker. Once we created an image, that image contains all the necessary thing which are required for running that specific application. Once image is created/ready we can run it anywhere.

1. **Multi-platform support**

Once you have a docker image installed, you can run the Docker image on a *local machine, or a data center, or inside the cloud*. Each of the cloud providers have amazing *support for Docker*. Amazon Web Services, Azure, Google Cloud Platform. All of these provide amazing options to run your containers, and they also provide communities implementations to do container orchestration, and run these containers. So the important thing to understand is the fact that once you build a Docker image, whether it's for a java application, whether it's a javascript application, or any other kind of application, you can run it on any of the platforms, on your local machine, inside a data center, or inside the cloud.

1. **Light weight and isolation**

**DOCKER IMAGES COMMANDS**

1. **Docker images** => show all the images
2. **Docker pull <image\_name>** => it will pull the specified image

This command will not create a container, it will just pull an image and make it available locally.

1. **Docker search <image\_name>** => to search available images
2. **Docker image inspect <imageId>** => we can see lot of details about the specific image.
3. **Docker image remove <imageId>** => remove the image

Sometimes a container is referencing the image then if you are going to remove the image, it may give an error. So in that case you need to remove the image first.

1. **Docker container rm ID** =>
2. **Docker container ls –a** => it gives you all the containers

When we are running above command, it shows all containers with their status. The container with status OK cannot be removed directly so we need to stop it with below command and then we can remove.

1. **Docker container stop <ID>** => to stop the container

**DOCKER CONTAINER COMMANDS**

1. **Docker container ls –a** => it gives you all the containers

When we are running above command, it shows all containers with their status. The container with status OK cannot be removed directly so we need to stop it with below command and then we can remove.

1. **Docker container stop <ID>** => to stop the container
2. **Docker container pause <container\_ID> =>** to pause a container
3. **Docker container unpause <container\_ID> =>** to unpause a container
4. **Docker container stop <container\_ID> =>** to stop a container