**DOCKER CONCEPTS**

## Installation commands

* In ubuntu 16.04, docker installation commands

**sudo apt-get -f install**

* Install some required packages on your system for installing Docker on Ubuntu system.

***sudo apt-get install apt-transport-https ca-certificates curl software-properties-common***

* Now import dockers official GPG key to verify packages signature before installing them with apt-get. Run the below command on terminal.

***curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -***

* After that add the Docker repository on your Ubuntu system which contains Docker packages including its dependencies. You must have to enable this repository to install Docker on Ubuntu.

***sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu xenial stable"***

***sudo apt-get update***

* Run the following

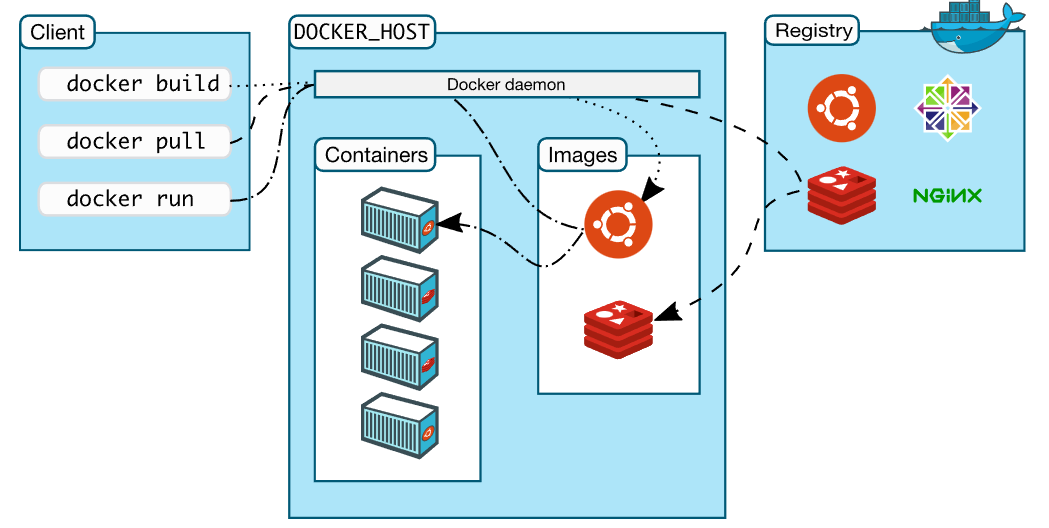
***apt-cache search docker-ce***

* sample output:

docker-ce - Docker: the open-source application container engine

* Install docker-ce:

***sudo apt-get install docker-ce***

******

## Docker Container Commands

Docker container is a running instance of an image. A container combined only libraries and settings required to make the application work. It is the lightweight and portable encapsulated environment for an application.

### Run Docker Container

* Use docker run command to launch a Docker container on your system. For example below command will create a Docker container from the hello-world image.

**docker run hello-world**

* Now create a real instance of Docker container using CentOS operating system. The option-it will provide an interactive session with pseudo-TTY enabled. This provides you container shell immediately.

**docker run -it centos**

* Use docker ps command to list all running containers on your current system. This will not list stopped containers. This will show you the Container ID, name and other use full information about the container.

**docker ps**

* Use -a option with above command to list all containers including stopped.

**docker ps -a**

* Find all Details of Container

Use docker inspect command to find all details about a docker container. You need to provide container id or container name to get details of the specific container.

**docker inspect <containerid>**

* Delete Docker Container

Use docker rm command to delete existing docker container. You need to provide container id or container name to delete specific container.

**docker stop <containerid>**

**docker rm <containerid>**

### Docker Images

An image is an inert, immutable, file that’s essentially a snapshot of a container. The images can be created with the build command, which can be used to create a container when started with run.

* List Docker Images. Use docker images command to list all images available on your local system.

**$ docker images**

* Download Docker Images

You use docker pull command to download any image from docker hub. For example download centos image from dockerhub to your local system and use latest to create containers.

**$ docker pull centos**

* Delete Docker Images

Use docker rmi command to remove any docker image from your local system. For example to remove image named centos use following command.

**$ docker rmi centos**

### Docker Data Volumes

* Sharing Host Volume to Docker Container
* You can also share data between the host machine and docker container. For example you can share /var/www directory of host machine to /data directory. You can share any directory of the host machine to a container.
* $ docker run -it -v /var/www:/data centos

#### Working with Dockerfile

**Dockerfile is a file used to build images by reading instructions from a file. The default name is used as Dockerfile. You can create dockerfile in the current directory with specific instructions and build a customized image as per your requirements.**

* Build Image with Dockerfile
* As a best practice, the Dockerfile is called Dockerfile and located in the root of the context. You can use the following command to build docker image. This will read Dockerfile in the current directory.

**$ docker build -t image\_name .**

* You can also use the -f flag with docker build command to point to a Dockerfile anywhere in your file system.

**$ docker build -t image\_name -f /path/to/Dockerfile .**

* What are Dockerfile Directives
* In our previous tutorial, you learned how to build images with Dockerfile. This tutorial will help you to understand the basic Dockerfile directives and there uses.

**FROM**

* The from directive is used to set base image for the subsequent instructions. A Dockerfile must have FROM directive with valid image name as the first instruction.
* Examples:

**FROM ubuntu**

**LABEL**

* Using label you can organize images in a proper way. this is useful to set maintainer address, vender name, version of image, release date etc. The line must be begain with keywork “LABEL”.

**LABEL maintainer="kiran"**

**LABEL vendor="org"**

**LABEL com.example.version="0.0.1"**

**RUN**

* Using RUN directing ,you can run any command to image during build time. For example you can install required packages during the build of image.

**RUN apt-get update**

**RUN apt-get install -y apache2 automake build-essential curl**

* As a more formatted syntax, you can use as following.

**RUN apt-get update && apt-get install -y \**

**automake \**

**build-essential \**

**curl \**

**COPY**

* The COPY directive used for coping files and directories from host system to the image during build. For example the first commands will copy all the files from hosts html/ directory /var/www/html image directory. Second command will copy all files with extension .conf to /etc/apache2/sites-available/ directory.

**COPY html/\* /var/www/html/**

**COPY \*.conf /etc/apache2/sites-available/**

WORKDIR

* The WORKDIR directive used to sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD commands during build.

**WORKDIR /opt**

**CMD**

* The CMD directive is used to run the service or software contained by your image, along with any arguments during the launching the container. CMD uses follwoing basic syntax

**CMD ["executable","param1","param2"]**

**CMD ["executable","param1","param2"]**

* For example, to start Apache service during launch of container, Use the following command.

**CMD ["apachectl", "-D", "FOREGROUND"]**

**EXPOSE**

* The EXPOSE directive indicates the ports on which a container will listen for the connections. After that you can bind host system port with container and use them.

**EXPOSE 80**

**EXPOSE 443**

**ENV**

* The ENV directive is used to set environment variable for specific service of container.

**ENV PATH=$PATH:/usr/local/pgsql/bin/ \**

**PG\_MAJOR=9.6.0**

**VOLUME**

* The VOLUME directive creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers.

**VOLUME ["/data"]**

If you are using Docker for development, you must be familiar with the -v or --volume flag that lets you mount your local files into the container. For instance, you can mount your local ./target onto the /usr/share/nginx/html directory container or an nginx container to visualize your html files.

echo "<h1>Hello from Host</h1>" > ./target/index.html

docker run -it --rm --name nginx -p 8080:80 -v "$(pwd)"/target:/usr/share/nginx/html nginx

### Manage Ports in Docker

The Docker containers run services inside it on a specific port. To access that services of container running on a port, You need to bind container port with some Docker host port.

### Docker POC’s

#### Running MySQL as Docker

* Have a look at below image. You will see that docker host is running two containers, one is running Apache which has some website and other have MySQL.
* MySQL Docker container

**docker pull mysql/mysql-server**

* If need to use version, run, inspect logs and port mapping

**docker pull mysql/mysql-server:5.5**

**docker run --name=mysql1 -d mysql/mysql-server**

**If need to use version**

**docker run --name=mysql1 -d mysql/mysql-server:5.5**

**docker logs mysql1**

**docker run --name=mysql1 -e MYSQL\_ROOT\_HOST=% -p 3306:3306 -d mysql/mysql-server**

**docker run --name=mysql1 -e MYSQL\_ROOT\_HOST=% -p 3306:3306 -d mysql/mysql-server:5.5**

**docker logs mysql1 2>&1 | grep GENERATED**

* This will give generated root password, provide access

docker exec -it mysql1 mysql -u root -p

**ALTER USER 'root'@'localhost' IDENTIFIED BY 'admin@123';**

* to check all the containers, include not-running containers

**docker ps -a**

**CREATE USER 'newUser'@'%' IDENTIFIED BY 'newPass123';**

# [How to resolve Unable to load authentication plugin 'caching\_sha2\_password' issue](https://stackoverflow.com/questions/50387952/how-to-resolve-unable-to-load-authentication-plugin-caching-sha2-password-issu)

**ALTER USER 'newUser'@'%' IDENTIFIED WITH mysql\_native\_password BY 'newPass123';**

**GRANT ALL PRIVILEGES ON \*.\* TO 'newUser'@'%' WITH GRANT OPTION;**

#### Run Tomcat as Docker

* Run the default Tomcat server (CMD ["catalina.sh", "run"]):

**$ docker run -it --rm tomcat:8.0**

* You can test it by visiting http://container-ip:8080 in a browser or, if you need access outside the host, on port 8888:

**$ docker run -it --rm -p 8888:8080 tomcat:8.0**

**-p <<host port>>:<<container port>>**

* You can then go to http://localhost:8888 or http://host-ip:8888 in a browser

*Build a war file and create a docker file to run the war file as a tomcat docker container*

*Docker file for custom war file*

**FROM tomcat:8.0.43-jre8**

**ADD webdemodocker.war /usr/local/tomcat/webapps/**

**EXPOSE 8080**

**CMD chmod +x /usr/local/tomcat/bin/catalina.sh**

**CMD ["catalina.sh", "run"]**

**sudo docker build -t webdemo .**

**sudo docker run -i -t -d -p 8080:8080 webdemo**

**Docker file for nginx**

***From nginx***

***COPY index.html /usr/share/nginx/html***

**index.html**

***This is hello world to Niginx***

**docker build -t mynginx .**

**docker run -it -p 8080:80 mynginx**

**docker run -it -p 8081:80 mynginx**

**Running 2 container instances on same port mapped to different host port**

**Create a spring boot application**

**Create a docker file**

***FROM openjdk:8-jdk-alpine***

***VOLUME /tmp***

***COPY ${JAR\_FILE} springbootapp.jar***

***ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","springbootapp.jar"]***

***EXPOSE 8080***

*If running****Java 8 on modern OSes with support to Deterministic Random Bit Generator (DRBG)****, I'd recommend using  
-Djava.security.egd=file:/dev/urandom to avoid getting the code blocked unexpectedly. If not sure about the OS being used, my suggestion is to stick with the original recommendation, namely:  
-Djava.security.egd=file:/dev/./urandom*

*If running****Java 11****, I'd recommend simply using  
-Djava.security.egd=file:/dev/./urandom to make sure of:*

1. *leveraging the strongest****SecureRandom****implementation available (DRBG) regardless the underpinning platform*
2. *avoiding getting the code blocked unexpectedly (securerandom.source=file:/dev/urandom)*

Copy the jar and dockerfile to one location

Build the docker image

sudo docker build -t kiranresearch2020/pochub:springbootapp --build-arg JAR\_FILE="springbootapp.jar" .

### The Docker daemon

The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

### The Docker client

The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

### Docker Desktop

Docker Desktop is an easy-to-install application for your Mac, Windows or Linux environment that enables you to build and share containerized applications and microservices. Docker Desktop includes the Docker daemon (dockerd), the Docker client (docker), Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper. For more information, see [Docker Desktop](https://docs.docker.com/desktop/).

### Docker registries

A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.

### Docker objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

#### Images

An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a Dockerfile with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.