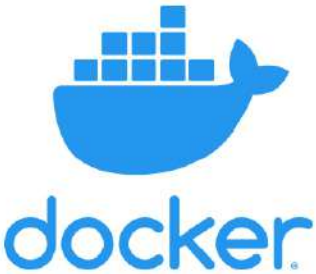
MANAGE DOCKER FILE



MAIL: Rnraju4u@gmail.com NUMBER: +91 9848363431 YOUTUBE: SYSGEEKS

DOCKER FILE

- Dockerfile is the core file that contains instructions to be performed when an image is built.
- It is a text document that contains all the commands a user could call on the CLI to assemble an image.
- The docker build command builds an image from a Dockerfile and a context.

SYNTAX: **Instruction** **arguments**

NOTE:

- In Dockerfile **"#"** is a single line comment.
- The escape character is used both to escape characters in a line, and to escape a newline. This allows a Dockerfile instruction to span multiple lines.
- Note that regardless of whether the escape parser directive is included in a Dockerfile, escaping is not performed in a RUN command, except at the end of a line.

BUILDING IMAGES USING DOCKERFILE:

- The docker build command builds Docker images from a Dockerfile and a "context".
- A build's context is the set of files located in the specified PATH or URL.
- The URL parameter can refer to three kinds of resources: Git repositories, pre-packaged tarball contexts and plain text files.

Build with PATH:

#docker build .

Build with URL:
#docker build github.com/creack/docker-firefox

Build with -:
#docker build - < Dockerfile

Build with Tag and File:

#docker build -f <path_to_Dockerfile> -t <REPOSITORY>:<TAG>

DOCKER FILE INSTRUCTIONS & ARGUMENTS

FROM:

- A docker file must start with a **FROM** instruction.
- The **FROM** instruction initializes a new build stage and sets the Base Image for subsequent instructions.

E.g.: FROM centos:latest

MAINTAINER:

- **MAINTAINER** is used to specify about the author who creates this new docker image for the support.

E.g.: MAINTAINER RAJU rnraju4u@gmail.com

LABEL:

- **LABEL** is used to specify metadata information to an Image, which is a **Key-Value Pair**.

E.g.: LABEL "App_Env"="Production"

RUN:

- It is used to executes any commands on top of the current image and this will create a new layer.
- It has two forms:

SHELL FORM:

E.g.: RUN yum update -y

EXECUTABLE FORM:

E.g.: RUN ["yum","update"]

CMD:

- It is used to set a command to be executed when running a container.
- There must be one CMD in a Dockerfile. If more than one CMD is listed, only the last CMD takes effect.

E.g.: CMD ping google.com

ENTRYPOINT:

- It is used to configure and run a container as an executable.

E.g.: ENTRYPOINT ping google.com

NOTE: If user specifies any arguments (commands) at the end of "docker run" command, the specified commands override the default in CMD instruction, But ENTRYPOINT instructional are not overwritten by the docker run command and ENTRYPOINT instruction will run as it is.

COPY:

- It is used to copy files, directories and Remote url files to the destination (Docker image) within the file system of the docker images.

E.g.: COPY src dest

NOTE: If the "src" argument is compressed file (tar, zip bzip2..etc), then it will copy exactly as a compressed file and will not extract.

ADD:

- It is used to copy files, directories and Remote URL files to the destination within the file system of the docker images.

E.g.: ADD src dest

NOTE: If the **"src"** argument is compressed file (tar, zip bzip2...etc), then it will Extract it automatically inside a destination in the Docker image.

WORKDIR:

- It is used to set the working directory.

  E.g.: WORKDIR /tmp

EXPOSE:

- This instruction informs Docker that the container listens on the specified network ports at runtime.
- By default, EXPOSE assumes TCP. You can also specify UDP.
- To publish the port when running the container, use the **-p** flag on docker run

  E.g.: EXPOSE 80/tcp

USER:

- The USER instruction lets you specify the username to be used when a command is run.
- It is used to set the user's name, group name, UID, GID for running subsequent commands. Else root user will be used.

  E.g.: USER webadmin

ENV:

- It is used to set environmental variables with **Key Value** set.

  E.g.: ENV username admin **(or) ENV** username=admin

ONBUILD:

- It lets you stash a set of commands that will be used when the image is used again as a base image for a container.

  E.g.: ONBUILD ADD

VOLUME:

- The VOLUME instruction is used to specify a mount point for a volume within the container.
- The volume will be created when the container is built, and it can be accessed and modified by processes running inside the container.

  E.g.: VOLUME /my_data

ARG:

- It is also used to set environment variables with Key-Value, but this variable will set only during the image build not on the container.

  E.g.: ARG tmp_ver 2.0

Understand how ARG and FROM interact:

FROM instructions support variables that are declared by any ARG instructions that occur before the first FROM.

```
ARG  CODE_VERSION=latest

FROM base:${CODE_VERSION}
```

E.g.:

```
ARG VERSION=latest

FROM centos:$VERSION

ARG VERSION

RUN echo $VERSION > image_version
```

PROJECT 1: SIMPLE DOCKERFILE IMAGE BUILD

```
FROM centos:7
MAINTAINER RAJU rnraju4u@gmail.com
RUN yum update -y
LABEL "Env"="Prod" \
      "Project"="Airtel" \
      "Version"="8.4"
COPY  *.txt  /opt/
```

PROJECT 2: BUILDING PYTHON FLASK

```
FROM centos:latest
MAINTAINER Raju "rnraju4u@gmail.com"
RUN yum update -y
RUN yum install -y python3 python3-pip wget
RUN pip3 install Flask
ADD hello.py /home/hello.py
WORKDIR /home
```

BUILDING DOCKER IMAGE FROM A FILE:

```
#docker build -t python3:latest .
#docker ps
#docker run -d -p 5000:5000 python:centos python3 hello.py
#docker logs cid
   Hello World
```

PROJECT 3: BUILDING APACHE HTTP SERVER

#Download an Image file

FROM centos:laest

#Installing Apache httpd on a image

RUN yum install httpd -y

#Copy index.html file into Document root location

COPY index.html /var/www/html/

#Running Port with 80

EXPOSE 80

#To make service to start

CMD ["/usr/sbin/httpd","-D","FOREGROUND"]

Create an index.html file in the locattion

#vim index.html

WELCOME TO SYSGEEKS…!

BUILDING DOCKER IMAGE FROM A FILE:

#docker build -t webserver:httpd-2.4 .

#docker run -d -it –name webserver -p 8000:80 webserver:httpd-2.4

#docker ps

Go To Web Browser, type:

http://10.10.10.10:8000

IMAGE FROM CONTAINER CHANGES

DOCKER COMMIT: Create a new image from a container's changes

- It can be useful to commit a container's file changes or settings into a new image. This allows you to debug a container by running an interactive shell, or to export a working dataset to another server.
- The commit operation will not include any data contained in volumes mounted inside the container.
- By default, the container being committed and its processes will be paused while the image is committed.

Download and Create a New Container:

#docker pull centos

#docker images

#docker run -d -it –name Mycentos centos

#docker ps

Connect and Changes on a Container:

#docker exec -it Mycentos bash

Changing Centos Repo mirrors in a container

#cd /etc/yum.repos.d/

#sed -i 's/mirrorlist/#mirrorlist/g' /etc/yum.repos.d/CentOS-*

#sed -i 's|#baseurl=http://mirror.centos.org|baseurl=http://vault.centos.org|g' /etc/yum.repos.d/CentOS-*

#yum update -y

#exit

Create an image from Container changes:

#docker commit CID My-Centos:raju

#docker image ls

MAIL: Rnraju4u@gmail.com **NUMBER:** +91 9848363431 **YOUTUBE:** SYSGEEKS

DOCKER HUB

- Docker Hub is a service provided by Docker for **finding and sharing container images** with your team.
- Link your images to the **GitHub/Bitbucket** repositories that can be built automatically based on web hooks.

PUBLIC REPOSITORIES:

- Users get access to free public repositories for storing and sharing images.
- Anyone can use **docker pull command** to download an image and run or build further images from it.

PRIVATE REPOSITORIES:

- Private repositories are just that private.
- Users can choose a **subscription plan** for private repositories.

DOCKER HUB Vs DOCKER SUBSCRIPTION

DOCKER HUB:

- Shareable image, but it can be private.
- No hassle of self-hosting.
- Free (except for a certain number of private images).

DOCKER SUBSCRIPTION:

- Integrated into your authentication services (that is, **AD/LDAP**).
- Deployed on your own infrastructure (or **cloud**).
- Commercial support.

NOTE: By default, repositories are pushed as public.

DOCKER HUB ENTERPRISE

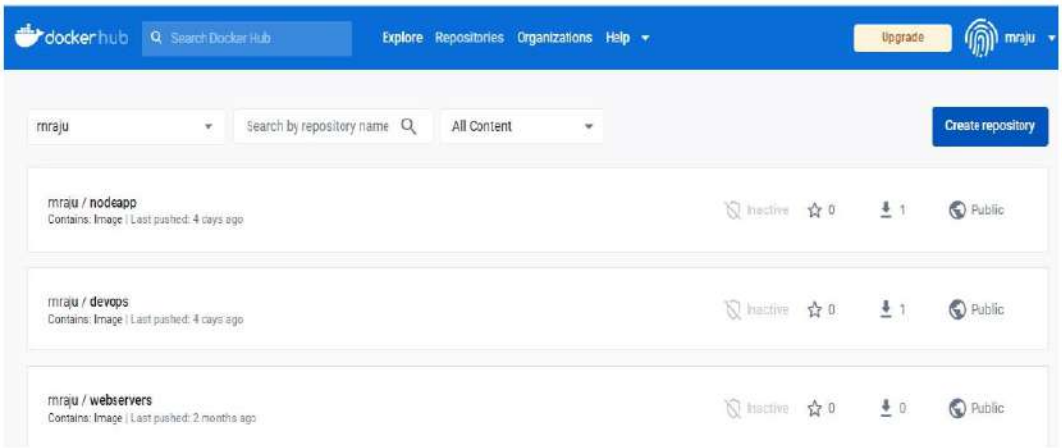
- It offers you is access to the software, access to **updates/patches/security fixes,** and support relating to issues with the software.
- The open-source Docker repository image doesn't offer these services at this level;

MANAGING DOCKER HUB:

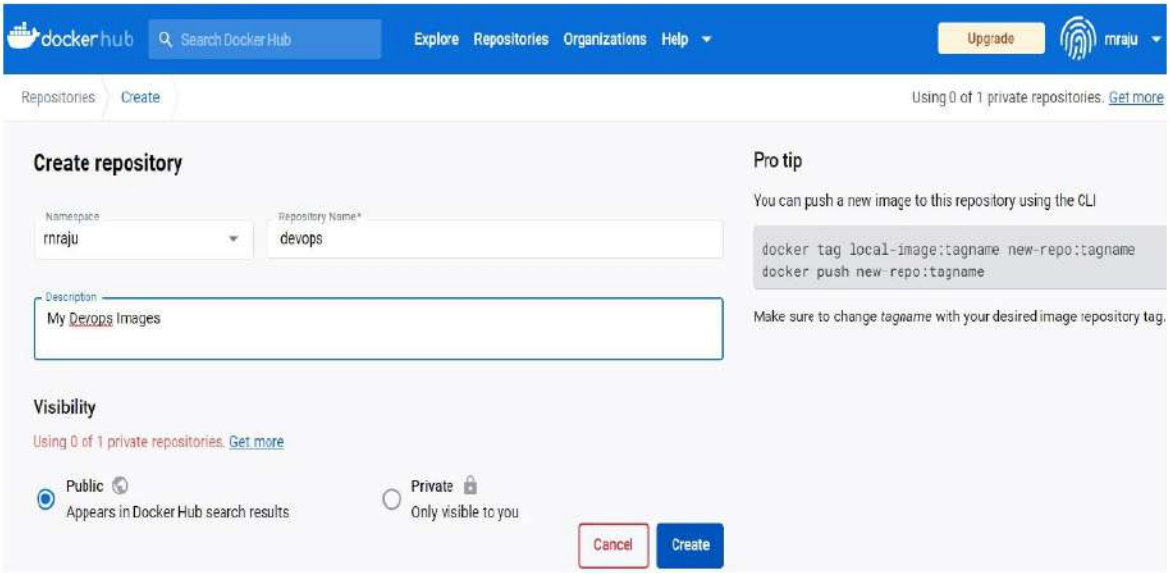
STEP 1: Sign up and Login docker hub through web interface:

https://hub.docker.com/

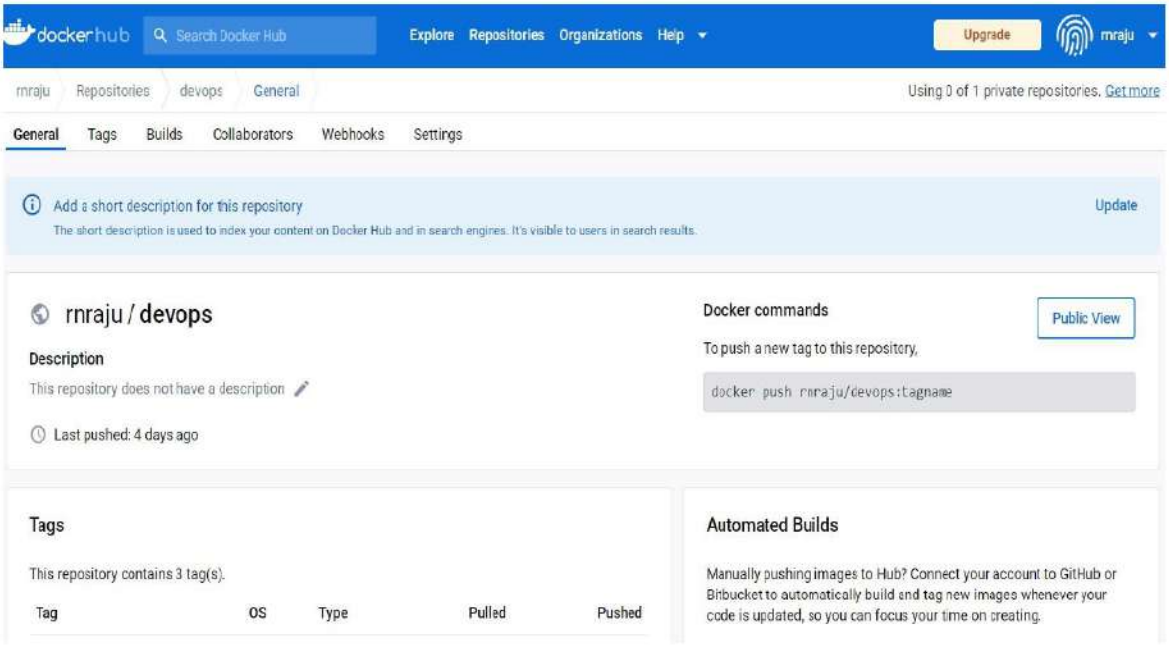
STEP 2: Select Create Repository:



Create a Repository:



STEP 3: Verify Created Repository



DOCKER HUB FROM THE CLI:

#docker login

Login : rnraju

Pass : xxxxx

BUILDING & PUSHING AN IMAGES TO DOCKER HUB:

PROJECT 1: BUILDING WEB SERVER IMAGE

STEP1: Create a Dockerfile

#vim Dockerfile

#Installing Apache Webserver

#From Ubuntu Image

FROM ubuntu:latest

#Update image

RUN apt-get update -y

#Installing Apache2 server

RUN apt-get install apache2 -y

#Copy index.html file into Document root location

COPY index.html /var/www/html/

#Running Port with 80

EXPOSE 80

#To make service to start

CMD ["apache2ctl", "-D", "FOREGROUND"]

STEP2: Building an Image:

#docker build -t rnraju/webserver:2.4 .

#docker images

STEP3: Make a container for Testing

#docker run -it -d -p 6000:80 --name Apache-webserver rnraju/webserver:2.4

#docker ps

STEP3: Go to Browser

http://IP-Address:6000

STEP4: Push the image to Docker hub

#docker login

#docker push rnraju/webserver:2.4

STEP5: Go and Verify in docker Hub

PROJECT 2: BUILDING DOCKER IMAGE FOR NODEJS:

STEP 1: Create a Dockerfile

#vim Dockerfile

#Specify a base image

FROM node:alpine

#Copy Local files to container

WORKDIR /usr/app

COPY ./ /usr/app

#Install Some Dependencies

RUN npm install

#Default Command

CMD ["npm", "start"]

STEP 2: Create a "package.json" file

#vim package.json

```
{
"dependencies": {
"express": "*"
},
"scripts": {
"start": "node index.js"
}
}
```

STEP 3: Create a "index.js" file

#vim index.js

```
const express = require('express');
const app = express();
app.get('/', (req, res) => {
res.send('WELCOME TO NODEJS...');
});
app.listen(8080, () => {
 console.log('Listening on port 8080');
});
```

STEP 4: Build an image:

#docker build -t rnraju/nodejs .

#docker images

STEP 5: Running a container from the image file

#docker run -d -it –name mynodejs -p 4000:8080 rnraju/nodejs

#docker images

STEP 6: Enter the container with shell prompt

#docker run -it rnraju/nodejs

#cd /usr/app

#ls

STEP 7: Go and connect to the browser:

http://IP-Address:4000

STEP 8: Push the image to Docker hub

#docker login

#docker push rnraju/nodejs

STEP 9: Go and Verify in Docker Hub Repository