

-d -detach from terminal & run

- # systemctl status docker.
- # systemctl restart docker.
- # systemctl enable docker ← enable the service in booting sequence.

### Docker Commands

1. run: Start a Container.

- # docker run image  
OR
- # docker run image:latest
- # docker run image:1.1.0

# docker run -d image:1.1.0  
→ run's in background.

-d -detach.

Search image in local first if not present it downloads from dockerhub.

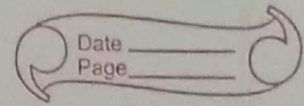
2. ps: list of running containers.

- # docker ps
- # docker ps -q [previously installed]
- # docker ps -q [quiet: list the id of container.]
- # docker ps -aq (container id).

running  
all the  
containers.



- It → interactive terminal.



3) stop : Stop a container.

# docker stop Name or container ID.

# docker start container\_ID.

stop multiple running containers in one go.

# docker stop \$(docker ps -pq).

4) rm : Remove a container.

# docker rm name/container ID

# docker rm -f name/container ID.

eg

# docker rm \$(docker ps -aq).

# docker rm -f \$(docker ps -aq).

5) images : list images.

# docker images.

# docker images -q.

6) rmi : Remove image [-f forcefully].

# docker rmi <image\_id>.

# docker rmi -f <image\_id>.

eg:

docker rmi \$(docker images -q)

docker rmi -f \$(docker images -q).

7) pull : Only download image  
 # docker pull image name.

8) exec - execute command.  
 # docker exec <name> cat /etc/passwd.  
 # docker exec -it <name/id> bash.  
 spawn a shell bash and  
 give interactive console.

9) Rename the container name.  
 docker run -d --name "Web1" httpd

10) Display the docker host information with  
 # docker info

Json format.

Array [ ].

Obj { }.

Key: Value..

NULL ← if no key & values present.

```
[ {
  "cdac": { "name": "    ",
            "class": "    "
          }
}
```

```
[ { "cdac": { "name": "    ", "class": "    "
              }
```



Date \_\_\_\_\_  
Page \_\_\_\_\_

# cat data.json | jq. → Dot represents first object and next should be key.

# cat data.json | jq '.cdac.harm'

```
graph LR; Root["# cat data.json | jq '.cdac.harm'"] --> cdac["cdac"]; Root --> harm["harm"]; cdac --> harm; cdac -- key --> K1[key]; harm -- key --> K2[key]; harm -- "1st Obj" --> O1[1st Obj];
```

# cat dot.json | jq '[].cdac\_love'

entire code      array

Docker inspect -f "{{ json.Id }}" container\_id.

# docker run -d --name "web1" -p 8080:80 httpd

publish.

host port container port image name

- m → module  
- t → tag

### Volume Mapping:

container httpd : /usr/local/apache2/htdocs

Host OS : /web1

```
# mkdir /web1
```

```
# cat > /web1/index.html
```

VLALA

```
# docker run -d --name "server1" -p 80:80 -v  
/web1:/usr/local/apache2/htdocs httpd:latest
```

- Create Acc on docker.

- Create repository

- Docker Push:

```
# docker login
```

username:

password:

FROM ubuntu

RUN apt update

RUN apt install python3 -y

COPY ./prog.py /mnt/prog.py

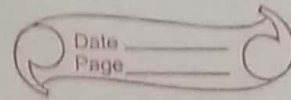
ENTRYPOINT python3 /mnt/prog.py

```
docker build -t repo/image name  
babbar/ops
```

```
docker push . repo/docker image name
```



- input from user and place.



```
copy ./index.html /usr/local/apache2/htdocs  
EXPOSE 80
```

## 2<sup>1</sup> Export Docker Images:

```
# docker ps
```

```
# docker export webapp1 > webapp.tar
```

You could commit this container as a new image locally, but you could also use the Docker Import.

Command:

```
# docker import - mywebapp < webapp.tar
```

View: `tar -tvf demo.tar`

Extract: `tar -xvf demo.tar -C demo`

↑  
extract  
view forcefully

↓  
extract in

## Docker Compose.

YAML:

1. [Key - value] pair:  
Key: value

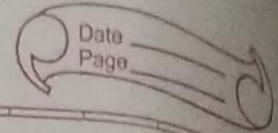
eg:

fruit : Apple

Veg : carrot.

Esc.

!q-exit  
:wq-save



2. Array/List :

Fruit :

- Orange
- Apple

3. Dict :

It a set properties that group together.

Banana :

Calories : 102

Grapes :

Calories : 99

[+] key value / dict / list :

Fruits :

- Banana :

Calories : 102

- Grapes :

Calories : 99

Install

apt install docker-compose -y

Make folder & write

Create a file named as docker-compose.yml

Services :

web :

image : "httpd:latest"

ports :

- "80:80"



volumes:

- /web1 : /usr/local/apache2/htdocs.

web2:

image : "nginx:latest"

ports:

- "80:80"

# docker-compose config ← verify syntax & yaml version.

# docker-compose up → foreground

# docker-compose up -d. ← Background

# docker-compose ps. ← composed To see list of docker compose.

# docker-compose down → container stop & remove.

★

version : '3.9'

services:

kido:

image : "python:3"

ports:

- "80:80"

command : python3 -m http.server 80

working\_dir : /mnt

volumes:

- /lol : /mnt



if config docker  
system restart docker.

vim Dockerfile.

FROM ubuntu

RUN apt update

~~ap~~ RUN apt install -y . . .

RUN pip3 install flask

RUN pip3 install flask-mysql.

COPY ./app.py ./

ENTRYPOINT

cat > app.py

import os.

~~./app.py~~ FLASK\_APP=./app.py flask

run -- host=0.0.0.0

host 0.0.0.0.)

docker build -t <sup>babadai</sup> ~~pgdai~~/cdac  
<sub>present</sub>

deploy

docker-compose -y ml

services:

cdac.dai:

build: .

image: babadai/cdac

ports:

- "5000 : 8080"

docker-compose up -d.  
" " ps

## Link Container:

By linking containers, you provide a secure channel via which Docker containers can communicate to each other.

First: NoSQL Data Structure Server Redis.

- # docker pull redis.
- # docker run -d --name db redis.
- # docker ps.

Second: Now let us run a another container, named as "ubuntu" container.

- # docker run -d --link db:redis --name myserver ubuntu:latest.
- # cat /etc/hosts.
- # ping redis.
- link flag is source containername:containeraliasname.