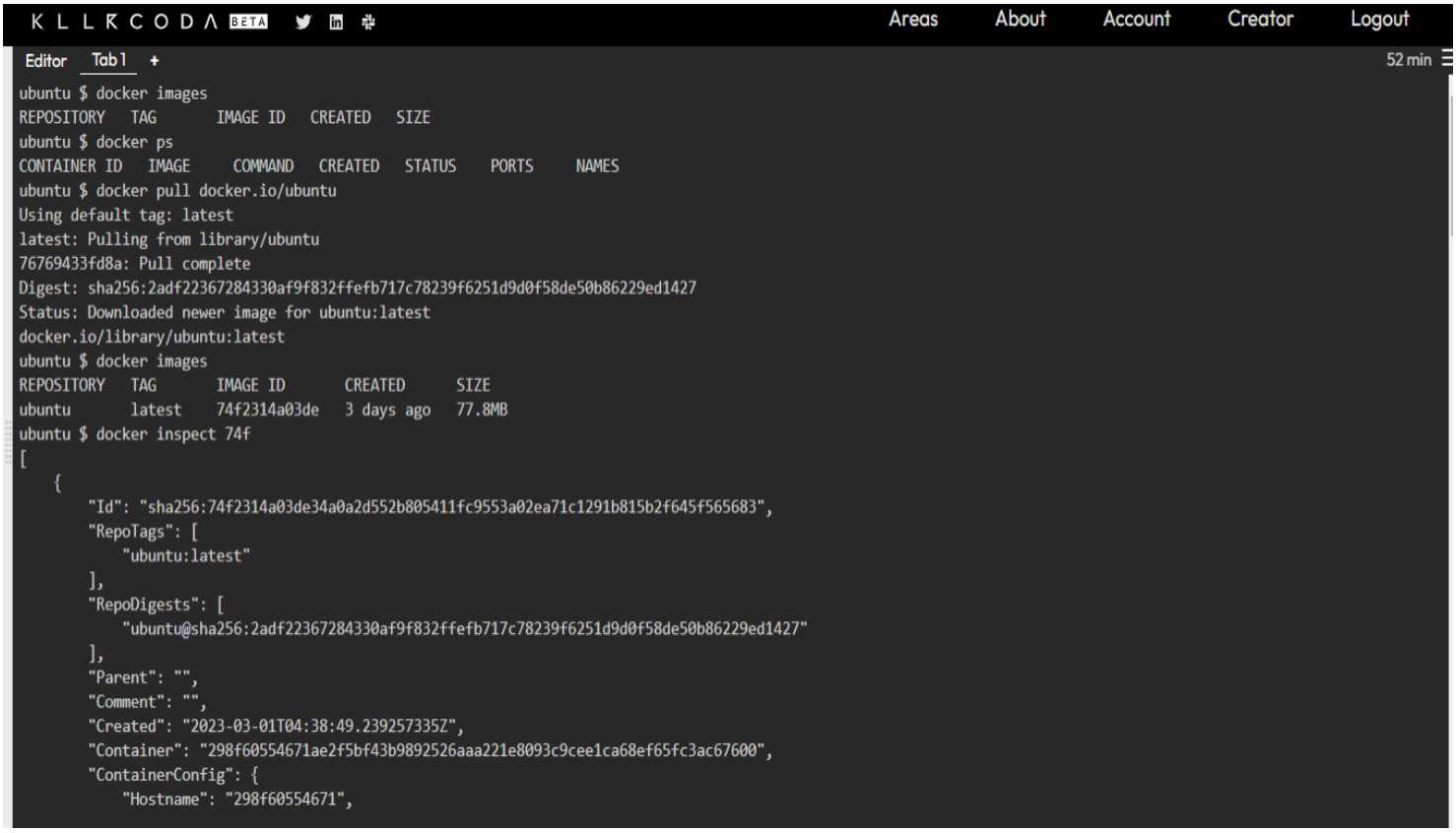
TABLE OF CONTENTS

DOCKER IMAGES

1. BASE IMAGE

a. **UBUNTU:**



Editor Tab 1 + 50 min

```
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED      SIZE
ubuntu       latest    74f2314a03de   3 days ago   77.8MB
ubuntu $ docker run -it ubuntu bash
root@7ef90ba5b73f:/# cat /etc/os
cat: /etc/os: No such file or directory
root@7ef90ba5b73f:/# cat /etc/os-release
PRETTY_NAME="Ubuntu 22.04.2 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.2 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=jammy
root@7ef90ba5b73f:/# docker ps
bash: docker: command not found
root@7ef90ba5b73f:/# exit
exit
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ubuntu $ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED              STATUS                       PORTS     NAMES
7ef90ba5b73f   ubuntu    "bash"    About a minute ago   Exited (127) 51 seconds ago            friendly_kalam
ubuntu $ docker start 7ef
7ef
```

```
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED         STATUS         PORTS     NAMES
7ef90ba5b73f   ubuntu    "bash"    2 minutes ago   Up 4 seconds             friendly_kalam
ubuntu $ docker exec -it 7ef bash
root@7ef90ba5b73f:/# touch file1
root@7ef90ba5b73f:/# ls
bin  boot  dev  etc  file1  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@7ef90ba5b73f:/# exit
exit
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED         STATUS         PORTS     NAMES
7ef90ba5b73f   ubuntu    "bash"    2 minutes ago   Up 46 seconds            friendly_kalam
ubuntu $ docker stop 7ef
7ef
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ubuntu $ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED         STATUS                     PORTS     NAMES
7ef90ba5b73f   ubuntu    "bash"    3 minutes ago   Exited (137) 7 seconds ago           friendly_kalam
```

B. CENTOS:

KLLRCODA BETA Areas About Account Creator Logout

Editor Tab 1 + 45 min

```
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED      SIZE
ubuntu       latest    74f2314a03de   3 days ago   77.8MB
ubuntu $ docker pull docker.io/centos
Using default tag: latest
latest: Pulling from library/centos
a1d0c7532777: Pull complete
Digest: sha256:a27fd8080b517143cbbbab9dfb7c8571c40d67d534bbdee55bd6c473f432b177
Status: Downloaded newer image for centos:latest
docker.io/library/centos:latest
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
ubuntu       latest    74f2314a03de   3 days ago      77.8MB
centos       latest    5d0da3dc9764   17 months ago   231MB
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ubuntu $ docker inspect 5d0
[
    {
        "Id": "sha256:5d0da3dc976460b72c77d94c8a1ad043720b0416bfc16c52c45d4847e53fadb6",
        "RepoTags": [
            "centos:latest"
        ],
        "RepoDigests": [
            "centos@sha256:a27fd8080b517143cbbbab9dfb7c8571c40d67d534bbdee55bd6c473f432b177"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2021-09-15T18:20:05.184694267Z",
        "Container": "9bf8a9e2ddff4c0d76a587c40239679f29c863a967f23abf7a5babb6c2121bf1",
```

KLLRCODA BETA Areas About Account Creator Logout

Editor Tab 1 + 44 min

```
ubuntu $ docker run -it centos bash
[root@54b53d5f31b1 /]# cat /etc/os-release
NAME="CentOS Linux"
VERSION="8"
ID="centos"
ID_LIKE="rhel fedora"
VERSION_ID="8"
PLATFORM_ID="platform:el8"
PRETTY_NAME="CentOS Linux 8"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:centos:centos:8"
HOME_URL="https://centos.org/"
BUG_REPORT_URL="https://bugs.centos.org/"
CENTOS_MANTISBT_PROJECT="CentOS-8"
CENTOS_MANTISBT_PROJECT_VERSION="8"
[root@54b53d5f31b1 /]# touch file2
[root@54b53d5f31b1 /]# ls
bin  dev  etc  file2  home  lib  lib64  lost+found  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
[root@54b53d5f31b1 /]# exit
exit
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ubuntu $ docker start 5d0
```

KLLRCODA BETA

Areas About Account Creator Logout

Editor Tab 1 +

43 min

```
ubuntu $ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED          STATUS                      PORTS     NAMES
54b53d5f31b1   centos    "bash"    59 seconds ago   Exited (0) 36 seconds ago             wizardly_knuth
7ef90ba5b73f   ubuntu    "bash"    11 minutes ago   Exited (137) 8 minutes ago            friendly_kalam
ubuntu $ docker start 54b
54b
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED             STATUS         PORTS     NAMES
54b53d5f31b1   centos    "bash"    About a minute ago   Up 3 seconds             wizardly_knuth
ubuntu $ docker exec -it 54b
"docker exec" requires at least 2 arguments.
See 'docker exec --help'.

Usage:  docker exec [OPTIONS] CONTAINER COMMAND [ARG...]

Run a command in a running container
ubuntu $ docker exec -it 54b bash
[root@54b53d5f31b1 /]# ls
bin  dev  etc  file2  home  lib  lib64  lost+found  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
[root@54b53d5f31b1 /]# exit
exit
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED             STATUS         PORTS     NAMES
54b53d5f31b1   centos    "bash"    About a minute ago   Up 29 seconds            wizardly_knuth
ubuntu $ docker stop 54b
54b
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

```
ubuntu $ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED             STATUS                      PORTS     NAMES
54b53d5f31b1   centos    "bash"    About a minute ago   Exited (0) 5 seconds ago             wizardly_knuth
7ef90ba5b73f   ubuntu    "bash"    12 minutes ago       Exited (137) 9 minutes ago           friendly_kalam
ubuntu $
```

2. BASE + APPLICATION PLATFORM:

a. PYTHON:

KLLRCODA BETA

Areas About Account Creator Logout

Editor Tab 1 +

35 min

```
ubuntu $ docker pull docker.io/python
Using default tag: latest
latest: Pulling from library/python
32fb02163b6b: Pull complete
167c7feebee8: Pull complete
d6dfff1f6f3d: Pull complete
e9cdcd4942eb: Pull complete
ca3bce705f6c: Pull complete
5e1c6c4f8bbf: Pull complete
efba3dc31239: Pull complete
b45fafb4411c: Pull complete
70eb3e954fe5: Pull complete
Digest: sha256:d3c16df33787f3d03b2e096037f6deb3c1c5fc92c57994a7d6f2de018de01a6b
Status: Downloaded newer image for python:latest
docker.io/library/python:latest
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
python       latest    33ce09363487   2 days ago      925MB
ubuntu       latest    74f2314a03de   3 days ago      77.8MB
centos       latest    5d0da3dc9764   17 months ago   231MB
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ubuntu $ docker run -it python
Python 3.11.2 (main, Mar  1 2023, 14:46:02) [GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
```

Editor Tab 1 +

34 min

```
>>> kd = "Hello Connections!! \nHope you all are in good health and spirit!! \nThis is my third docker image : python."
>>> print(kd)
Hello Connections!!
Hope you all are in good health and spirit!!
This is my third docker image : python.
```

```
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ubuntu $ docker ps -a
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS                       PORTS     NAMES
4db56ae0d77c   python    "python3"   3 minutes ago    Exited (0) 41 seconds ago              nervous_bhabha
54b53d5f31b1   centos    "bash"      11 minutes ago   Exited (0) 9 minutes ago               wizardly_knuth
7ef90ba5b73f   ubuntu    "bash"      21 minutes ago   Exited (137) 18 minutes ago            friendly_kalam
ubuntu $ docker start 4db
4db
ubuntu $ docker ps -a
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS                       PORTS     NAMES
4db56ae0d77c   python    "python3"   3 minutes ago    Up 2 seconds                           nervous_bhabha
54b53d5f31b1   centos    "bash"      11 minutes ago   Exited (0) 9 minutes ago               wizardly_knuth
7ef90ba5b73f   ubuntu    "bash"      22 minutes ago   Exited (137) 18 minutes ago            friendly_kalam
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND     CREATED          STATUS         PORTS     NAMES
4db56ae0d77c   python    "python3"   3 minutes ago    Up 7 seconds             nervous_bhabha
```

3. BASE + APPLICATION PLATFORM + CODE:

a. HTTPD:

KLLRCODA BETA Areas About Account Creator Logout

Editor Tab 1 + 28 min

```
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
python       latest    33ce09363487   2 days ago      925MB
ubuntu       latest    74f2314a03de   3 days ago      77.8MB
centos       latest    5d0da3dc9764   17 months ago   231MB
ubuntu $ docker pull httpd
Using default tag: latest
latest: Pulling from library/httpd
3f9582a2cbe7: Pull complete
9423d69c3be7: Pull complete
d1f584c02b5d: Pull complete
8f73a485e312: Pull complete
b7697f8af320: Pull complete
Digest: sha256:83e99e7c437898cb564bbd3ceba7f1ea3f2d86e1cbd7a5324940086e59082f2b
Status: Downloaded newer image for httpd:latest
docker.io/library/httpd:latest
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
python       latest    33ce09363487   2 days ago      925MB
httpd        latest    b304753f3b6e   3 days ago      145MB
ubuntu       latest    74f2314a03de   3 days ago      77.8MB
centos       latest    5d0da3dc9764   17 months ago   231MB
ubuntu $ docker run -d httpd
22aa3b99f88683ae9b40d16cfc140910db61b6cfd068950b2a6584ae29a2eb97
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED         STATUS         PORTS     NAMES
22aa3b99f886   httpd     "httpd-foreground"   3 seconds ago   Up 3 seconds   80/tcp    kind_greider
```

Editor Tab 1 +


```
            "Gateway": "172.17.0.1",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "172.17.0.2",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "MacAddress": "02:42:ac:11:00:02",
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID": "db8bda17d0c7f051e6895bf43ab457989fd28abd9d52d46ebc9c018397d18957",
                    "EndpointID": "3f162ab8d3042764997dc267a2d7dbe46dc53956042cf6e94b860583a6b3abe5",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:02",
                    "DriverOpts": null
                }
            }
        }
    }
]
ubuntu $ curl 172.17.0.2
<html><body><h1>It works!</h1></body></html>
```

b. OPENSHIFT:

KLLKCODA BETA Areas About Account Creator

Editor Tab 1 +

```
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
python       latest    33ce09363487   2 days ago      925MB
httpd        latest    b304753f3b6e   3 days ago      145MB
ubuntu       latest    74f2314a03de   3 days ago      77.8MB
centos       latest    5d0da3dc9764   17 months ago   231MB
ubuntu $ docker pull docker.io/openshift/hello-openshift
Using default tag: latest
latest: Pulling from openshift/hello-openshift
4f4fb700ef54: Pull complete
8b32988996c5: Pull complete
Digest: sha256:aaea76ff622d2f8bcb32e538e7b3cd0ef6d291953f3e7c9f556c1ba5baf47e2e
Status: Downloaded newer image for openshift/hello-openshift:latest
docker.io/openshift/hello-openshift:latest
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED         STATUS         PORTS     NAMES
22aa3b99f886   httpd     "httpd-foreground"   4 minutes ago   Up 4 minutes   80/tcp    kind_greider
ubuntu $ docker images
REPOSITORY                 TAG       IMAGE ID       CREATED         SIZE
python                     latest    33ce09363487   2 days ago      925MB
httpd                      latest    b304753f3b6e   3 days ago      145MB
ubuntu                     latest    74f2314a03de   3 days ago      77.8MB
centos                     latest    5d0da3dc9764   17 months ago   231MB
openshift/hello-openshift  latest    7af3297a3fb4   4 years ago     6.09MB
ubuntu $ docker run -d openshift/hello-openshift
4aaf6e9a951c151a8294b87838c7c4451b495fc076dcd8dcc975061bc0969e06
ubuntu $ docker ps
CONTAINER ID   IMAGE                      COMMAND              CREATED          STATUS          PORTS                NAMES
4aaf6e9a951c   openshift/hello-openshift  "/hello-openshift"   4 seconds ago    Up 4 seconds    8080/tcp, 8888/tcp   jovial_taussig
22aa3b99f886   httpd                      "httpd-foreground"   5 minutes ago    Up 5 minutes    80/tcp               kind_greider
ubuntu $
```

KLLKCODA BETA Areas About Account Creator

Editor Tab 1 +

```
ubuntu $ docker ps
CONTAINER ID   IMAGE                      COMMAND              CREATED              STATUS              PORTS                NAMES
4aaf6e9a951c   openshift/hello-openshift  "/hello-openshift"   About a minute ago   Up About a minute   8080/tcp, 8888/tcp   jovial_taussig
22aa3b99f886   httpd                      "httpd-foreground"   6 minutes ago        Up 6 minutes        80/tcp               kind_greider
ubuntu $ curl 172.17.0.3
curl: (7) Failed to connect to 172.17.0.3 port 80: Connection refused
ubuntu $ curl 172.17.0.3:8080
Hello OpenShift!
ubuntu $ curl 172.17.0.3:8888
Hello OpenShift!
ubuntu $
```

Date: 4/3/23
Saturday

DIRECTLY RUNNING A CONTAINER, WITHOUT PULLING THE IMAGE:

```
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ubuntu $ docker run -d httpd
Unable to find image 'httpd:latest' locally
latest: Pulling from library/httpd
3f9582a2cbe7: Pull complete
9423d69c3be7: Pull complete
d1f584c02b5d: Pull complete
8f73a485e312: Pull complete
b7697f8af320: Pull complete
Digest: sha256:83e99e7c437898cb564bbd3ceba7f1ea3f2d86e1cbd7a5324940086e59082f2b
Status: Downloaded newer image for httpd:latest
8934b555a89057b6371d67f6eb2c07034c4310d8e7d8466ef3ca8b1f7e498135
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED          STATUS          PORTS     NAMES
8934b555a890   httpd     "httpd-foreground"   18 seconds ago   Up 17 seconds   80/tcp    friendly_kalam
ubuntu $
```

By default, the container is assigned a **fancy name**, for example, here **: friendly_kalam.**

NAMING A CONTAINER:

```
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
ubuntu $ docker run --name=kasturi -d httpd
Unable to find image 'httpd:latest' locally
latest: Pulling from library/httpd
3f9582a2cbe7: Pull complete
9423d69c3be7: Pull complete
d1f584c02b5d: Pull complete
8f73a485e312: Pull complete
b7697f8af320: Pull complete
Digest: sha256:83e99e7c437898cb564bbd3ceba7f1ea3f2d86e1cbd7a5324940086e59082f2b
Status: Downloaded newer image for httpd:latest
079603160c6821b128e335fd14709a2b7bb0afb3f370ed76ceefa04b8ff24e19
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED              STATUS              PORTS     NAMES
079603160c68   httpd     "httpd-foreground"   About a minute ago   Up About a minute   80/tcp    kasturi
```

Changing the content inside the default landing page: htdocs/index.html

```
ubuntu $ docker exec -it kasturi bash
root@06e0d7a518d9:/usr/local/apache2# cat htdocs/index.html
<html><body><h1>It works!</h1></body></html>
root@06e0d7a518d9:/usr/local/apache2# cd htdocs/
root@06e0d7a518d9:/usr/local/apache2/htdocs# echo "Kab hai Holi?????" > htdocs/index.html
bash: htdocs/index.html: No such file or directory
root@06e0d7a518d9:/usr/local/apache2/htdocs# echo "Kab hai Holi?????" > index.html
root@06e0d7a518d9:/usr/local/apache2/htdocs# cd ..
root@06e0d7a518d9:/usr/local/apache2# cat htdocs/index.html
Kab hai Holi?????
root@06e0d7a518d9:/usr/local/apache2# exit
exit
ubuntu $ curl 172.17.0.2
Kab hai Holi?????
ubuntu $
```

Problem that arises due to Naming:

```
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS     NAMES
06e0d7a518d9   httpd     "httpd-foreground"   5 minutes ago   Up 5 minutes   80/tcp    kasturi
ubuntu $ docker stop kasturi
kasturi
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ubuntu $ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS                     PORTS     NAMES
06e0d7a518d9   httpd     "httpd-foreground"   5 minutes ago   Exited (0) 5 seconds ago             kasturi
ubuntu $ docker run --name=kasturi -d openshift/hello-openshift
Unable to find image 'openshift/hello-openshift:latest' locally
latest: Pulling from openshift/hello-openshift
4f4fb700ef54: Pull complete
8b32988996c5: Pull complete
Digest: sha256:aaea76ff622d2f8bcb32e538e7b3cd0ef6d291953f3e7c9f556c1ba5baf47e2e
Status: Downloaded newer image for openshift/hello-openshift:latest
docker: Error response from daemon: Conflict. The container name "/kasturi" is already in use by container "06e0d7a518d9973c374dbc65e31b6cf455ffb4c007e118103d427eea
65aa688". You have to remove (or rename) that container to be able to reuse that name.
See 'docker run --help'.
ubuntu $
```

Even if we stop the earlier container, kasturi, it won't be deleted. So, whenever we try creating a new container with the same name, it will give back an error i.e. to either rename or to remove the container.

RENAMING THE CONTAINER:

```
ubuntu $ docker ps -a
CONTAINER ID   IMAGE     COMMAND              CREATED          STATUS                      PORTS     NAMES
06e0d7a518d9   httpd     "httpd-foreground"   10 minutes ago   Exited (0) 4 minutes ago              kasturi
ubuntu $ docker rename kasturi kd
ubuntu $ docker ps -a
CONTAINER ID   IMAGE     COMMAND              CREATED          STATUS                      PORTS     NAMES
06e0d7a518d9   httpd     "httpd-foreground"   10 minutes ago   Exited (0) 5 minutes ago              kd
ubuntu $ docker run --name=kasturi -d openshift/hello-openshift
6085139a1907ed732fbecd8f4726c96236aedc82b24d69625fae98a6d491d412
ubuntu $ docker ps
CONTAINER ID   IMAGE                         COMMAND             CREATED         STATUS         PORTS                NAMES
6085139a1907   openshift/hello-openshift     "/hello-openshift"  4 seconds ago   Up 3 seconds   8080/tcp, 8888/tcp   kasturi
ubuntu $
```

REMOVING THE CONTAINER:

```
ubuntu $ docker ps -a
CONTAINER ID   IMAGE                         COMMAND             CREATED             STATUS                      PORTS                NAMES
6085139a1907   openshift/hello-openshift     "/hello-openshift"  About a minute ago  Up About a minute           8080/tcp, 8888/tcp   kasturi
06e0d7a518d9   httpd                         "httpd-foreground"  12 minutes ago      Exited (0) 7 minutes ago                         kd
ubuntu $ docker rm kd
kd
ubuntu $ docker ps -a
CONTAINER ID   IMAGE                         COMMAND             CREATED         STATUS         PORTS                NAMES
6085139a1907   openshift/hello-openshift     "/hello-openshift"  2 minutes ago   Up 2 minutes   8080/tcp, 8888/tcp   kasturi
ubuntu $
```

ENVIRONMENT VARIABLE : OS-LEVEL

```
ubuntu $ env
SHELL=/bin/bash
PWD=/root
LOGNAME=kc-internal
HOME=/root
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
SSH_CONNECTION=10.48.1.23 39462 172.30.1.2 22
LESSCLOSE=/usr/bin/lesspipe %s %s
TERM=xterm-256color
LESSOPEN=| /usr/bin/lesspipe %s
USER=kc-internal
SHLVL=3
PS1=\h $
SSH_CLIENT=10.48.1.23 39462 22
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/go/bin:/snap/bin:/usr/local/go/bin:/snap/bin
MAIL=/var/mail/kc-internal
DEBIAN_FRONTEND=noninteractive
OLDPWD=/opt
_=/usr/bin/env
ubuntu $ echo $PWD
/root
```

```
ubuntu $ export NAME=KASTURI
ubuntu $ echo $NAME
KASTURI
ubuntu $ env
SHELL=/bin/bash
NAME=KASTURI
PWD=/root
LOGNAME=kc-internal
HOME=/root
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=
01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.
zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz
=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.c
ab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01
;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.
```

ENVIRONMENT VARIABLE IN CONTAINER: (-e)

```
ubuntu $ docker run --name=myapp -e NAME=KASTURI -it ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
76769433fd8a: Pull complete
Digest: sha256:2adf22367284330af9f832ffefb717c78239f6251d9d0f58de50b86229ed1427
Status: Downloaded newer image for ubuntu:latest
root@75e3ff7daa5d:/# docker ps
bash: docker: command not found
root@75e3ff7daa5d:/# env
HOSTNAME=75e3ff7daa5d
NAME=KASTURI
PWD=/
HOME=/root
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;
01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;
zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31
=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01
ab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mj
;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;
m2v=01;35:*.mkv=01;35:*.webm=01;35:*.webp=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.
5:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.
;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;3
s=00;36:*.spx=00;36:*.xspf=00;36:
TERM=xterm
SHLVL=1
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
_=/usr/bin/env
root@75e3ff7daa5d:/# echo $NAME
KASTURI
root@75e3ff7daa5d:/#
```

MYSQL container: Live example of -e

```
ubuntu $ docker run --name=mydb -e MYSQL_ROOT_PASSWORD=kdkdkdkd -d mysql:5.6
Unable to find image 'mysql:5.6' locally
5.6: Pulling from library/mysql
35b2232c987e: Pull complete
fc55c00e48f2: Pull complete
0030405130e3: Pull complete
e1fef7f6a8d1: Pull complete
1c76272398bb: Pull complete
f57e698171b6: Pull complete
f5b825b269c0: Pull complete
dcb0af686073: Pull complete
27bbfeb886d1: Pull complete
6f70cc868145: Pull complete
1f6637f4600d: Pull complete
Digest: sha256:20575ecebe6216036d25dab5903808211f1e9ba63dc7825ac20cb975e34cfcae
Status: Downloaded newer image for mysql:5.6
c603d8e49df928735fa1fc9f68c573035392390089b77629994579a947ce51d9
ubuntu $ docker exec -it mydb bash
root@c603d8e49df9:/# mysql -uroot -pkdkdkdkd
Warning: Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.6.51 MySQL Community Server (GPL)

Copyright (c) 2000, 2021, Oracle and/or its affiliates. All rights reserved.
```

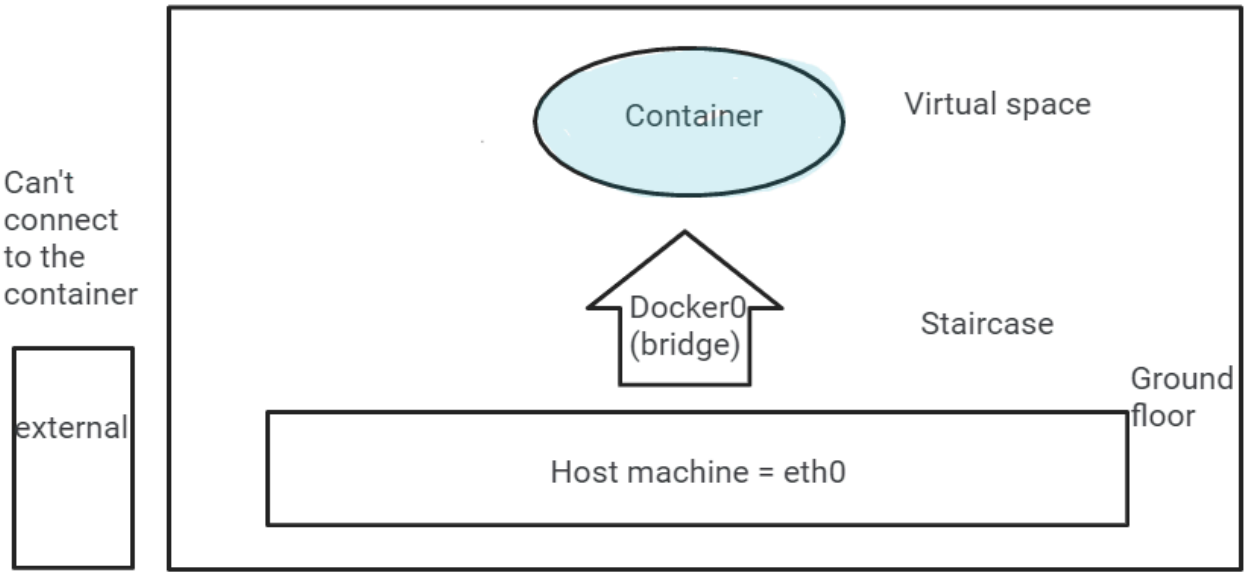
In the above screenshot, a password was assigned to the mysql container, with the help of -e.

But, if we don't give the password, then the container doesn't really run. It gives errors, which can be viewed in the logs.

```
ubuntu $ docker ps
CONTAINER ID   IMAGE                      COMMAND                  CREATED          STATUS          PORTS                    NAMES
c603d8e49df9   mysql:5.6                  "docker-entrypoint.s…"   2 minutes ago    Up 2 minutes    3306/tcp                 mydb
cdbdd538a5d5   httpd                      "httpd-foreground"       13 minutes ago   Up 13 minutes   80/tcp                   kd
6085139a1907   openshift/hello-openshift  "/hello-openshift"       16 minutes ago   Up 16 minutes   8080/tcp, 8888/tcp       kasturi
ubuntu $ docker run --name=mydb1 -d mysql:5.6
0551bb8bb23cb653d9de172ecfd8275f3fbccc9910474e8c5ec8f6ccb3f10458
ubuntu $ docker ps
CONTAINER ID   IMAGE                      COMMAND                  CREATED          STATUS          PORTS                    NAMES
c603d8e49df9   mysql:5.6                  "docker-entrypoint.s…"   3 minutes ago    Up 3 minutes    3306/tcp                 mydb
cdbdd538a5d5   httpd                      "httpd-foreground"       14 minutes ago   Up 14 minutes   80/tcp                   kd
6085139a1907   openshift/hello-openshift  "/hello-openshift"       17 minutes ago   Up 17 minutes   8080/tcp, 8888/tcp       kasturi
ubuntu $ docker exec -it mydb1 bash
Error response from daemon: Container 0551bb8bb23cb653d9de172ecfd8275f3fbccc9910474e8c5ec8f6ccb3f10458 is not running
ubuntu $ docker logs mydb1
2023-03-04 18:41:15+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 5.6.51-1debian9 started.
2023-03-04 18:41:15+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
2023-03-04 18:41:15+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 5.6.51-1debian9 started.
2023-03-04 18:41:15+00:00 [ERROR] [Entrypoint]: Database is uninitialized and password option is not specified
    You need to specify one of the following:
    - MYSQL_ROOT_PASSWORD
    - MYSQL_ALLOW_EMPTY_PASSWORD
    - MYSQL_RANDOM_ROOT_PASSWORD
ubuntu $
```

Date: 6th March 2023

Note: Inspect will never show the environment variables, it will only show the Passive Information.



Host machine connected to the container through a bridge : Docker0

Host / local Machine is connected to our Container through a Bridge / Staircase, namely, Docker0.

Entering the MYSQL container : exec:

```
ubuntu $ docker run --name=mydb1 -e MYSQL_ROOT_PASSWORD=kdkdkd -e MYSQL_DATABASE=kasturi -d my
sql:5.6
6c55818af7114b6ca438318d022750c8699c400d83b0171de1bcdcc944ac2c53
ubuntu $ docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED         STATUS         PORTS
NAMES
6c55818af711   mysql:5.6   "docker-entrypoint.s…"   3 seconds ago   Up 2 seconds   3306/tcp
mydb1
```

```
root@6c55818af711:/# mysql -uroot -pkdkdkd
Warning: Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.6.51 MySQL Community Server (GPL)

Copyright (c) 2000, 2021, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| kasturi            |
| mysql              |
| performance_schema |
+--------------------+
```

```
mysql> create database kd
    -> ;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| kasturi            |
| kd                 |
| mysql              |
| performance_schema |
+--------------------+
5 rows in set (0.00 sec)
```

#Connecting the host / local machine with the MYSQL container: -h : host

```
ubuntu $ docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED         STATUS         PORTS      NAMES
6c55818af711   mysql:5.6   "docker-entrypoint.s…"   5 minutes ago   Up 5 minutes   3306/tcp   mydb1
ubuntu $ sudo apt update -y
Hit:1 http://archive.ubuntu.com/ubuntu focal InRelease
Hit:2 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:4 http://archive.ubuntu.com/ubuntu focal-backports InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
95 packages can be upgraded. Run 'apt list --upgradable' to see them.
ubuntu $ sudo apt install mysql-client
Reading package lists... Done
Building dependency tree
Reading state information... Done
mysql-client is already the newest version (8.0.32-0ubuntu0.20.04.2).
0 upgraded, 0 newly installed, 0 to remove and 95 not upgraded.
ubuntu $
```

```
ubuntu $ mysql -h172.17.0.2 -uroot -pkdkdkd
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.6.51 MySQL Community Server (GPL)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| kasturi            |
| kd                 |
| mysql              |
| performance_schema |
```

Limitations of Docker:

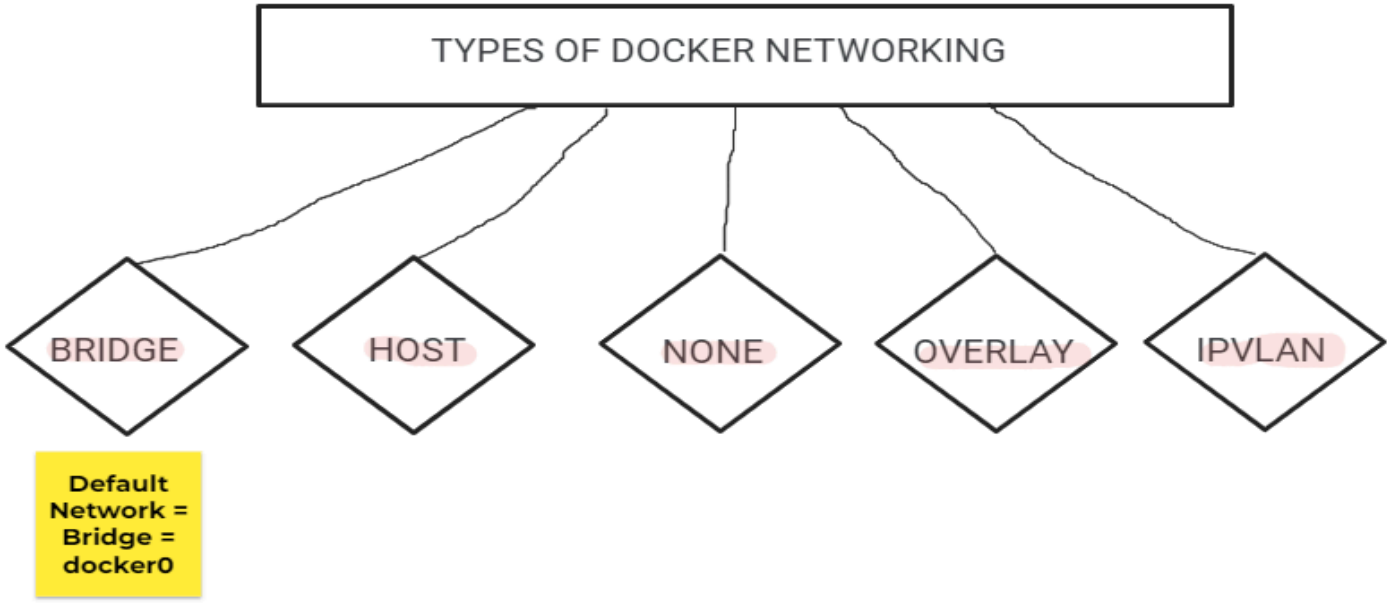
1. Docker containers are always **ISOLATED**.
2. We can't access the container from the **EXTERNAL** scenario.
3. Docker image is **IMMUTABLE** (which can't be changed).
4. Docker containers are **NON-PERSISTENT** (Removing the container will result in removal of all the data inside it.)

Date: 8th March 2023.

DOCKER NETWORKING

Features:
- Security (Because, by default, the container is isolated.)
- Portability (Because container is lightweight)
- Lightweight
- Upgradation is easy.
- Scalable
- Multi-tenancy* (conditions applied)

Docker Networking:
Docker networking refers to the process of connecting Docker containers and allowing them to communicate with each other or with external networks.



Bridge networking is the default networking mode in Docker, which creates a virtual network bridge between the Docker host and the containers, that allows containers to communicate with each other as well as with external networks.

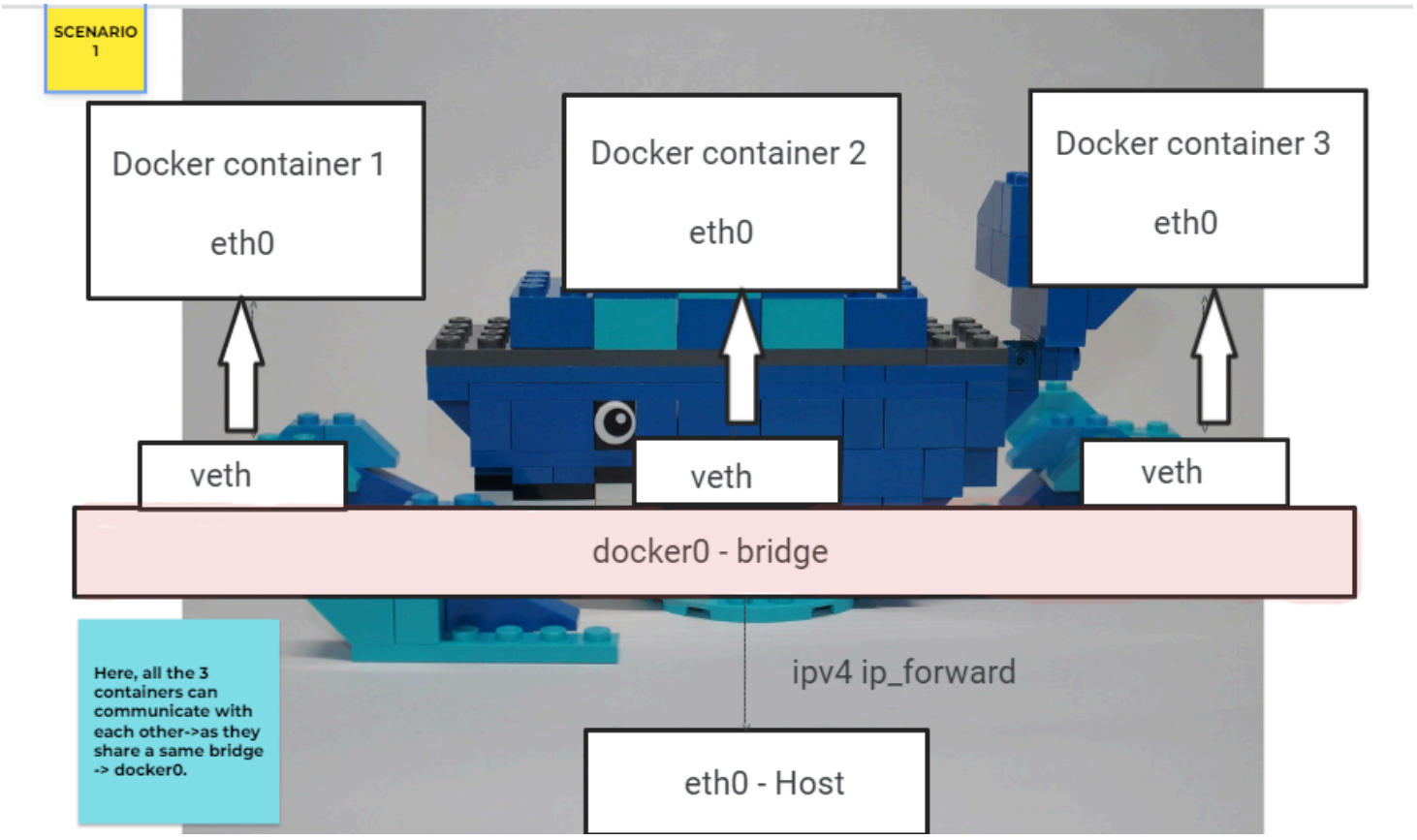
IP Forwarding:
IP forwarding is the process of forwarding network traffic from one network interface to another. IP forwarding is used to allow containers to communicate with external networks.

By default, IP forwarding is **disabled** in Docker containers for security reasons.
To enable IP forwarding in Docker, we need to set the **ipv4 ip forward** parameter to **1**. Thus the connection between eth0 (host) and docker0 (bridge) is possible.

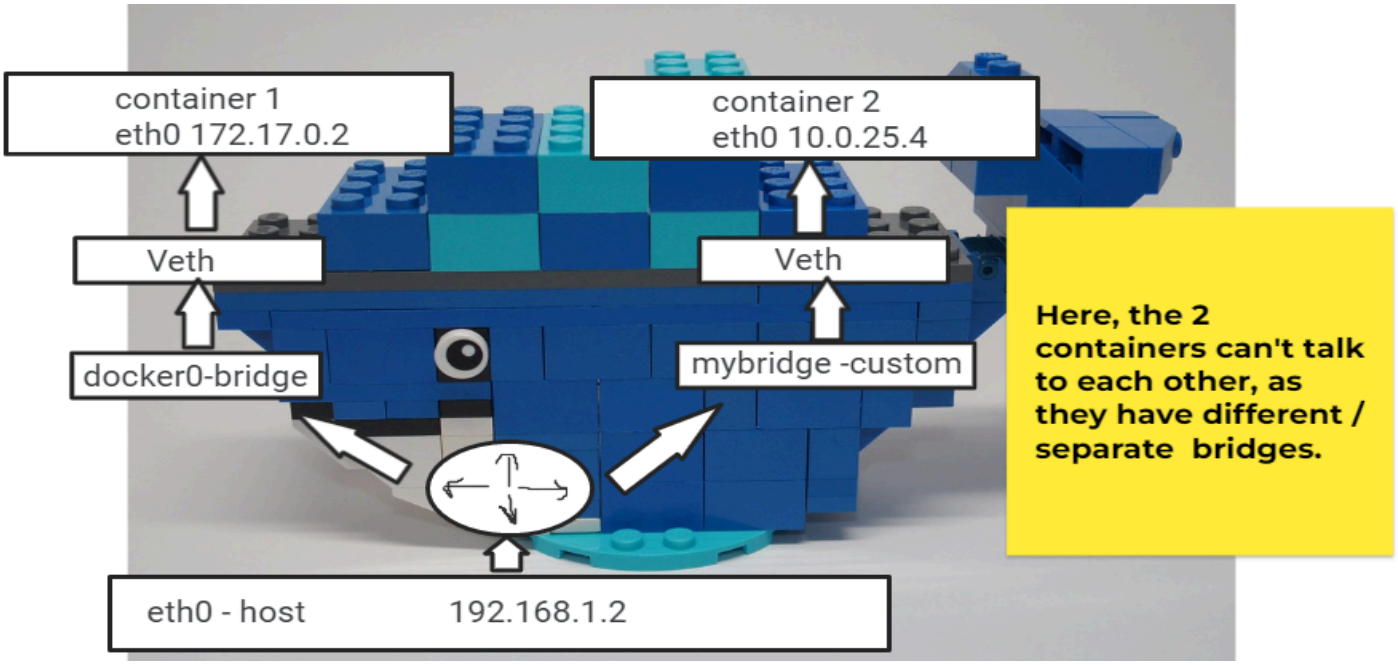
Virtual Ethernet Cable (Veth):

It connects the eth0 (container) with the docker0 bridge. It basically acts like a socket.

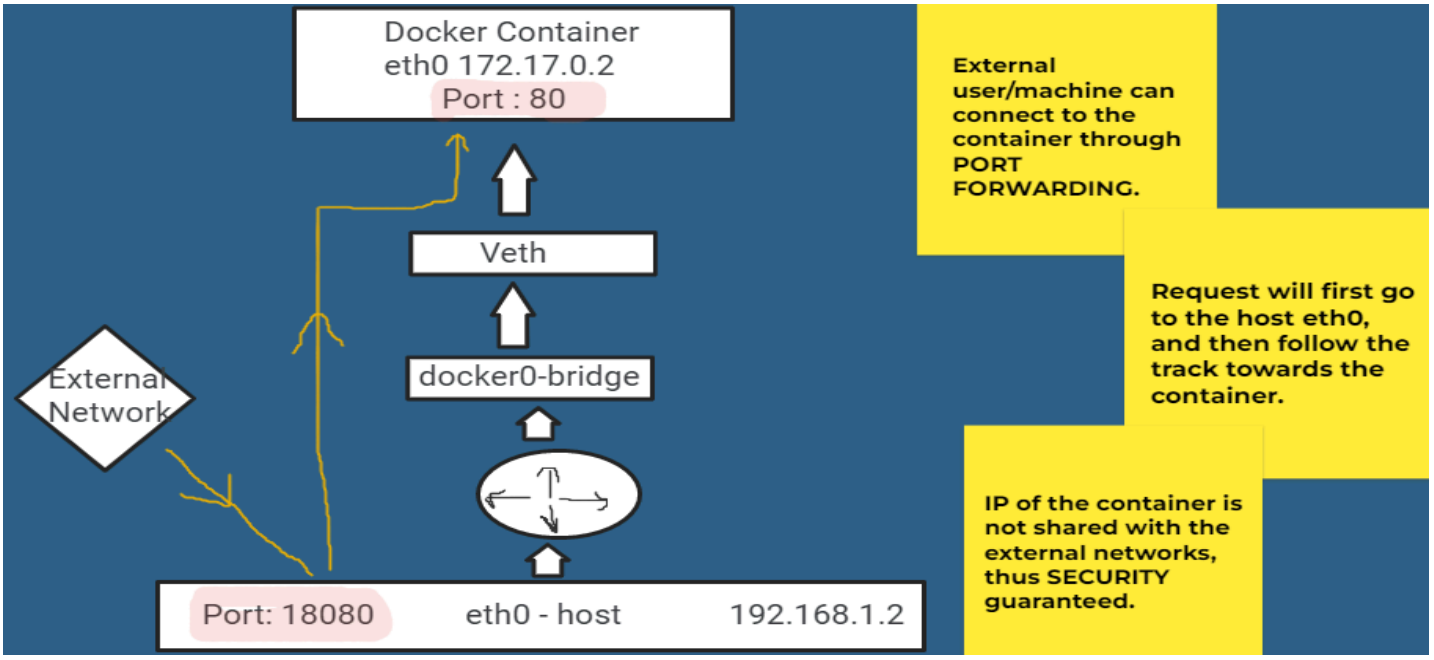
SCENARIO 1



SCENARIO 2



NAT (NETWORK ADDRESS TRANSLATION) / IP PORT FORWARDING

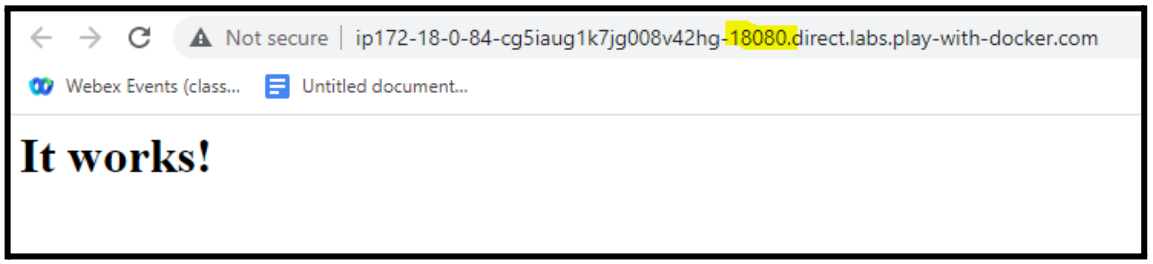


Command for Port Forwarding:
We can use the parameter -p and then add the source and destination ports.
= **docker run - - name=container_name -p host_port:container_port -d docker_image**

Note: DHCP (Dynamic Host Control Protocol) gives the IP address to a Container.

Practical: docker.io/httpd image

```
[node1] (local) root@192.168.0.28 ~
$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[node1] (local) root@192.168.0.28 ~
$ docker run --name=kasturi1 -p 18080:80 -d docker.io/httpd
Unable to find image 'httpd:latest' locally
latest: Pulling from library/httpd
3f9582a2cbe7: Pull complete
9423d69c3be7: Pull complete
d1f584c02b5d: Pull complete
758a20a64707: Pull complete
08507f82f391: Pull complete
Digest: sha256:76618ddd53f315a1436a56dc84ad57032e1b2123f2f6489ce9c575c4b280c4f4
Status: Downloaded newer image for httpd:latest
0f9fb49ce7ed60d5b9491492131d95948876811951f42e49dd8f9e0afb7c537c
[node1] (local) root@192.168.0.28 ~
$ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED         STATUS         PORTS                     NAMES
0f9fb49ce7ed   httpd     "httpd-foreground"   4 seconds ago   Up 3 seconds   0.0.0.0:18080->80/tcp     kasturi1
```



docker.io/openshift/hello-openshift image

```
[node1] (local) root@192.168.0.28 ~
$ docker run --name kasturi2 -p 28080:8080 -d docker.io/openshift/hello-openshift
Unable to find image 'openshift/hello-openshift:latest' locally
latest: Pulling from openshift/hello-openshift
4f4fb700ef54: Pull complete
8b32988996c5: Pull complete
Digest: sha256:aaea76ff622d2f8bcb32e538e7b3cd0ef6d291953f3e7c9f556c1ba5baf47e2e
Status: Downloaded newer image for openshift/hello-openshift:latest
2dd047ed2714af1a048ab72dac5ce1b9cd7be3e838b9df96f58a3fddba325891
[node1] (local) root@192.168.0.28 ~
$ docker ps
CONTAINER ID   IMAGE                      COMMAND              CREATED         STATUS         PORTS
                NAMES
2dd047ed2714   openshift/hello-openshift   "/hello-openshift"   5 seconds ago   Up 3 seconds   8888/tcp, 0.0.0.0:
28080->8080/tcp   kasturi2
0f9fb49ce7ed   httpd                      "httpd-foreground"   3 minutes ago   Up 3 minutes   0.0.0.0:18080->80/
tcp               kasturi1
```

docker.io/pengbai/docker-supermario

```
[node1] (local) root@192.168.0.28 ~
$ docker ps
[node1] (local) root@192.168.0.28 ~
$ docker run --name=kasturi3 -p 38080:8080 -d docker.io/pengbai/docker-supermario
Unable to find image 'pengbai/docker-supermario:latest' locally
latest: Pulling from pengbai/docker-supermario
092586df9206: Pull complete
ef599477fae0: Pull complete
4530c6472b5d: Pull complete
d34d61487075: Pull complete
```

Not secure | ip172-18-0-84-cg5iaug1k7jg008v42hg-38080.direct.labs.play-with-docker.com

Webex Events (class... Untitled document...

NETWORKING IN DETAIL

LISTING ALL DOCKER NETWORKS

```
ubuntu $ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
db8bda17d0c7   bridge    bridge    local
a4ed26aa0e2d   host      host      local
790fa6da653d   none      null      local
```

RUNNING 3 CONTAINERS ON SAME NETWORK : BRIDGE

1.Container 1 : httpd

```
ubuntu $ docker run --name=test1 -d docker.io/httpd
Unable to find image 'httpd:latest' locally
latest: Pulling from library/httpd
3f9582a2cbe7: Pull complete
9423d69c3be7: Pull complete
d1f584c02b5d: Pull complete
758a20a64707: Pull complete
08507f82f391: Pull complete
Digest: sha256:76618ddd53f315a1436a56dc84ad57032e1b2123f2f6489ce9c575c4b280c4f4
Status: Downloaded newer image for httpd:latest
6b1c638395dcd98e3cfca4e6f4c8ef4ea62da7298c26ba0562b9a661db9951aa
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED         STATUS         PORTS     NAMES
6b1c638395dc   httpd     "httpd-foreground"   3 seconds ago   Up 2 seconds   80/tcp    test1
```

2. Container 2: openshift

```
ubuntu $ docker run --name=test2 -d docker.io/openshift/hello-openshift
Unable to find image 'openshift/hello-openshift:latest' locally
latest: Pulling from openshift/hello-openshift
4f4fb700ef54: Pull complete
8b32988996c5: Pull complete
Digest: sha256:aaea76ff622d2f8bcb32e538e7b3cd0ef6d291953f3e7c9f556c1ba5baf47e2e
Status: Downloaded newer image for openshift/hello-openshift:latest
3c50b7824948cd33b47530ff18160c5c6e501f22114bf809e781657fd24d1f7d
ubuntu $ docker ps
CONTAINER ID   IMAGE                      COMMAND              CREATED             STATUS             PORTS
      NAMES
3c50b7824948   openshift/hello-openshift  "/hello-openshift"   3 seconds ago       Up 2 seconds       8080/tcp, 8888/
tcp   test2
6b1c638395dc   httpd                      "httpd-foreground"   About a minute ago  Up About a minute  80/tcp
      test1
```

3. Container 3: busybox (Busybox image contains almost all Linux commands)

```
ubuntu $ docker run --name=test3 -it docker.io/busybox sh
/ #
```

Trying to Ping the Container 1 and container 2:

IP Address of Container 1 i.e **test1**	IP Address of Container 2 i.e. **test2**
`"IPAddress": "172.17.0.2",`	`"IPAddress": "172.17.0.3",`

```
ubuntu $ docker run --name=test3 -it docker.io/busybox sh
/ #
/ # ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.319 ms
64 bytes from 172.17.0.2: seq=1 ttl=64 time=0.067 ms
64 bytes from 172.17.0.2: seq=2 ttl=64 time=0.065 ms
64 bytes from 172.17.0.2: seq=3 ttl=64 time=0.064 ms
^C
--- 172.17.0.2 ping statistics ---
```

```
/ # ping 172.17.0.3
PING 172.17.0.3 (172.17.0.3): 56 data bytes
64 bytes from 172.17.0.3: seq=0 ttl=64 time=0.087 ms
64 bytes from 172.17.0.3: seq=1 ttl=64 time=0.063 ms
64 bytes from 172.17.0.3: seq=2 ttl=64 time=0.074 ms
64 bytes from 172.17.0.3: seq=3 ttl=64 time=0.080 ms
^C
--- 172.17.0.3 ping statistics ---
```

This proves that Containers on the same network can **talk to each other**.

INSPECTING THE DEFAULT NETWORK: BRIDGE

```
ubuntu $ docker network inspect bridge
```

Here, we can get all the details about the network like subnet, gateway, etc.

```
"Config": [
    {
        "Subnet": "172.17.0.0/16",
        "Gateway": "172.17.0.1"
    }
```

We can view how many containers are running on this network:

```
"Containers": {
    "3c50b7824948cd33b47530ff18160c5c6e501f22114bf809e781657fd24d1f7d": {
        "Name": "test2",
        "EndpointID": "09876b21c773cc33bc770e23aa2b81409814571bf37201eda72bbc2fc091d666",
        "MacAddress": "02:42:ac:11:00:03",
        "IPv4Address": "172.17.0.3/16",
        "IPv6Address": ""
    },
    "6b1c638395dcd98e3cfca4e6f4c8ef4ea62da7298c26ba0562b9a661db9951aa": {
        "Name": "test1",
        "EndpointID": "8f104d4fda52a0dbf4bd0af5ca6222207483e028a6330b2d6ebec911b78946b0",
        "MacAddress": "02:42:ac:11:00:02",
        "IPv4Address": "172.17.0.2/16",
        "IPv6Address": ""
    }
```

CREATING A NETWORK [- - NETWORK = " NETWORK_NAME"]

```
ubuntu $ docker network create mynet1
8f1252f7478fbbe6d2be80558d5f7e11c4d082e0afd90c8b38030a50dbced087
ubuntu $ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
db8bda17d0c7   bridge    bridge    local
a4ed26aa0e2d   host      host      local
8f1252f7478f   mynet1    bridge    local
790fa6da653d   none      null      local
```

INSPECTING THE NETWORK : MYNET1

```
ubuntu $ docker network inspect mynet1
[
    {
        "Name": "mynet1",
        "Id": "8f1252f7478fbbe6d2be80558d5f7e11c4d082e0afd90c8b38030a50dbced087",
        "Created": "2023-03-15T12:03:42.662151586Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
```

RUNNING A CONTAINER ON OUR NETWORK : MYNET1

```
ubuntu $ docker run --name=container1 --network=mynet1 -it docker.io/busybox sh
/ #
```

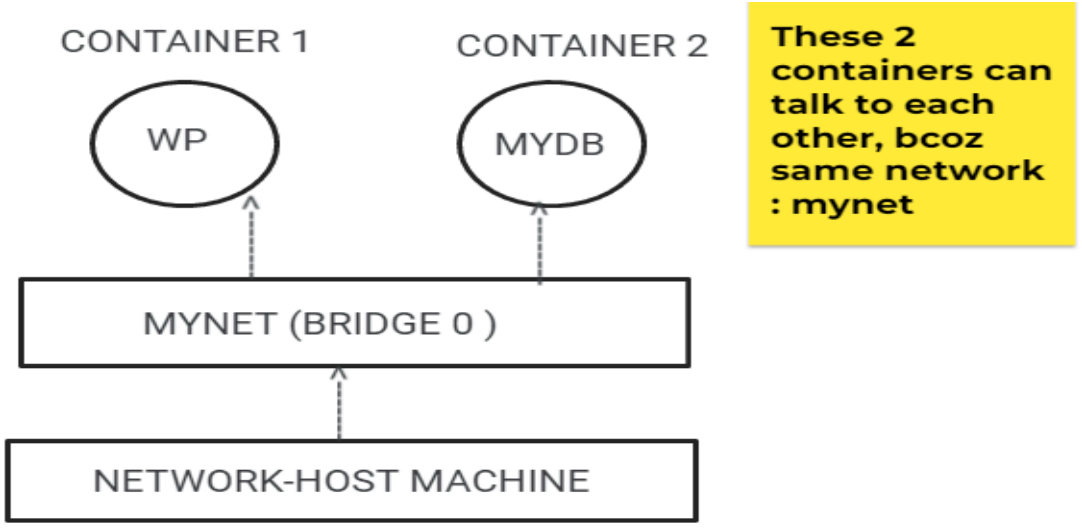
RUNNING ANOTHER CONTAINER ON THE SAME NETWORK

```
ubuntu $ docker run --name=container2 --network=mynet1 -d docker.io/httpd
675cc5652719ea937950dc94798154bed79e50efe980f80a63d0480d13733927
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED         STATUS         PORTS     NAMES
675cc5652719   httpd     "httpd-foreground"   5 seconds ago   Up 4 seconds   80/tcp    container2
```

TRYING TO PING THE CONTAINER2 FROM CONTAINER1: *USING NAME*

```
ubuntu $ docker run --name=container1 --network=mynet1 -it docker.io/busybox sh
/ #
/ # ping container2
PING container2 (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.198 ms
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.092 ms
64 bytes from 172.18.0.2: seq=2 ttl=64 time=0.066 ms
```

LAUNCHING MULTI - TIER APPLICATION (FRONTEND + BACKEND)



1. CREATING NETWORK

```
ubuntu $ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
db8bda17d0c7   bridge    bridge    local
a4ed26aa0e2d   host      host      local
790fa6da653d   none      null      local
ubuntu $ docker network create mynet1
2cad20f6308759b5fe79ce9407e937c03c874870aea2d3b383e96360d7421e48
ubuntu $ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
db8bda17d0c7   bridge    bridge    local
a4ed26aa0e2d   host      host      local
2cad20f63087   mynet1    bridge    local
790fa6da653d   none      null      local
```

2. CREATING BACKEND CONTAINER FIRST (DATABASE = MYDB)

Here, we created the container on our network=mynet1

```
ubuntu $ docker run --name=mydb --network=mynet1 -e MYSQL_ROOT_PASSWORD=centos -e MYSQL_DATABASE=kasturi -d mysql:5.6
Unable to find image 'mysql:5.6' locally
5.6: Pulling from library/mysql
35b2232c987e: Pull complete
fc55c00e48f2: Pull complete
0030405130e3: Pull complete
e1fef7f6a8d1: Pull complete
1c76272398bb: Pull complete
f57e698171b6: Pull complete
f5b825b269c0: Pull complete
dcb0af686073: Pull complete
27bbfeb886d1: Pull complete
6f70cc868145: Pull complete
1f6637f4600d: Pull complete
Digest: sha256:20575ecebe6216036d25dab5903808211f1e9ba63dc7825ac20cb975e34cfcae
Status: Downloaded newer image for mysql:5.6
e266c904e4648a48996b02db221cdc30c8aee5cebb15f820ca2bd8fcaf499633
ubuntu $ docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED         STATUS         PORTS      NAMES
e266c904e464   mysql:5.6   "docker-entrypoint.s…"   8 seconds ago   Up 7 seconds   3306/tcp   mydb
```

3. CREATING FRONTEND CONTAINER (WORDPRESS)

Note: Environment variable of wordpress image:

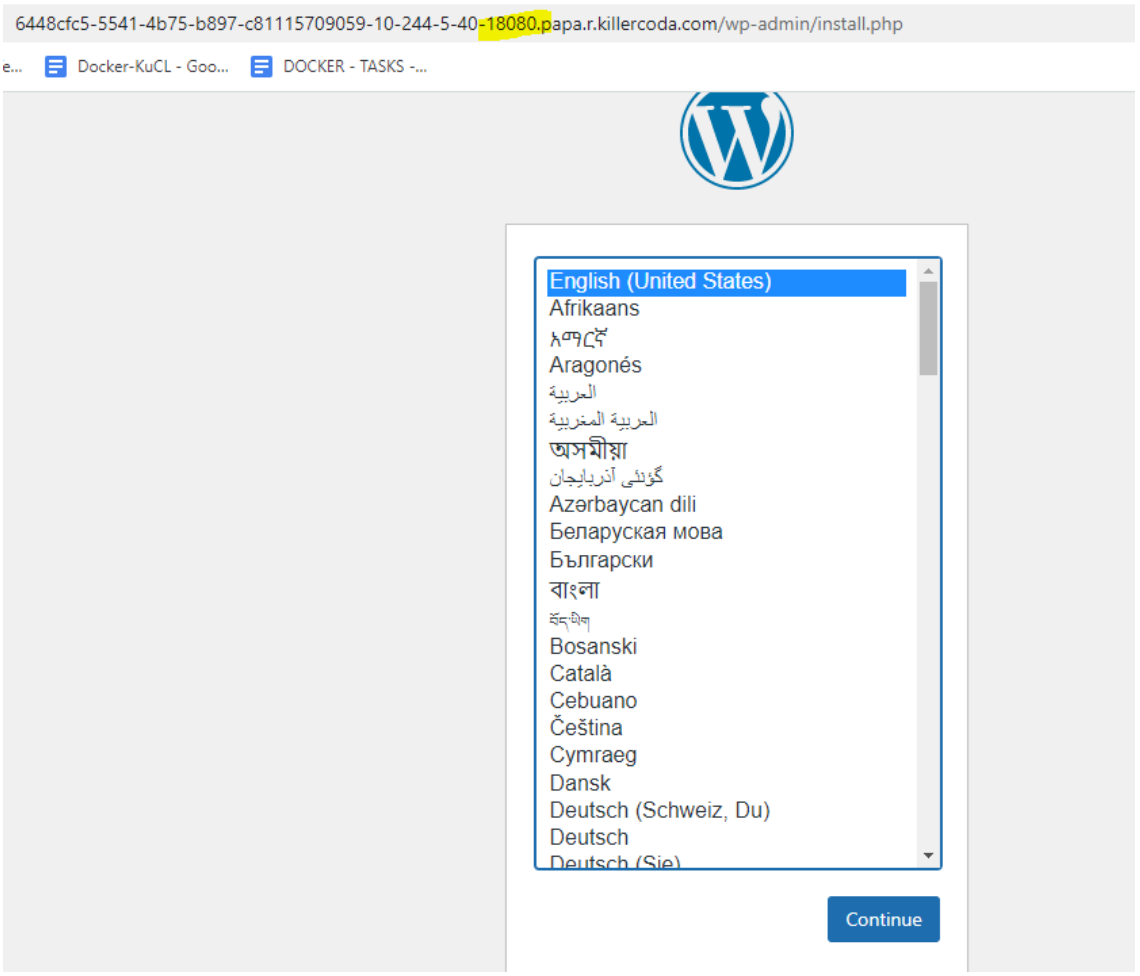
- `-e WORDPRESS_DB_HOST=...`
- `-e WORDPRESS_DB_USER=...`
- `-e WORDPRESS_DB_PASSWORD=...`
- `-e WORDPRESS_DB_NAME=...`
- `-e WORDPRESS_TABLE_PREFIX=...`
- `-e WORDPRESS_AUTH_KEY=...`, `-e WORDPRESS_SECURE_AUTH_KEY=...`, `-e WORDPRESS_LOGGED_IN_KEY=...`, `-e WORDPRESS_NONCE_KEY=...`, `-e WORDPRESS_AUTH_SALT=...`, `-e WORDPRESS_SECURE_AUTH_SALT=...`, `-e WORDPRESS_LOGGED_IN_SALT=...`, `-e WORDPRESS_NONCE_SALT=...` (default to unique random SHA1s, but only if other environment variable configuration is provided)
- `-e WORDPRESS_DEBUG=1` (defaults to disabled, non-empty value will enable `WP_DEBUG` in `wp-config.php`)
- `-e WORDPRESS_CONFIG_EXTRA=...` (defaults to nothing, non-empty value will be embedded verbatim inside `wp-config.php` -- especially useful for applying extra configuration values this image does not provide by default such as `WP_ALLOW_MULTISITE`; see docker-library/wordpress#142 for more details)

```
ubuntu $ docker run --name=wp --network=mynet1 -e WORDPRESS_DB_HOST=mydb -e WORDPRESS_DB_NAME=kasturi  -e WORDPRESS_DB_U
SER=root -e WORDPRESS_DB_PASSWORD=centos -p 18080:80 -d docker.io/wordpress
Unable to find image 'wordpress:latest' locally
latest: Pulling from library/wordpress
3f9582a2cbe7: Extracting [=========>                                         ]  6.226MB/31.41MB
0b95dc92ce55: Download complete
```

We will be mentioning the container mydb name in wordpress_db_host.

```
ubuntu $ docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS          PORTS
       NAMES
aaa5a5fd20e6   wordpress   "docker-entrypoint.s…"   28 seconds ago   Up 26 seconds   0.0.0.0:18080->80/tcp, :::18080->80
/tcp   wp
e266c904e464   mysql:5.6   "docker-entrypoint.s…"   4 minutes ago    Up 4 minutes    3306/tcp
       mydb
```

4. OPEN THE PORT: 18080



Continue, and fill up the information.

Welcome

Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.

Information needed

Please provide the following information. Do not worry, you can always change these settings later.

Site Title Kasturi A. Ghadge

Username kashdeshmukh

Usernames can have only alphanumeric characters, spaces, underscores, hyphens, periods, and the @ symbol.

Password TEpzU1e8AjB4nzaErY Hide

Strong

Important: You will need this password to log in. Please store it in a secure location.

Your Email kd@gmail.com

Double-check your email address before continuing.

Search engine visibility Discourage search engines from indexing this site

It is up to search engines to honor this request.

Log in
Go to Pages => Add New => Create a Website => Publish

KASTURI ARVIND GHADGE

Hello Connections!!
Greetings of the day!!

Welcome to QUOTES KI DUNIYA!!
I am a huge fan of JOHN M. GREEN, an American author.
Here is a short collection of quotes from a great Philanthropist.
Hope you enjoy!

"Sometimes, you read a book and it fills you with this weird evangelical zeal, and you become convinced that the shattered world will never be put back together unless and until all living humans read the book."
— **John Green,** The Fault in Our Stars

"You don't get to choose if you get hurt in this world...but you do have some say in who hurts you. I like my choices."
— **John Green,** The Fault in Our Stars

"Some infinities are bigger than other infinities."
— **John Green,** The Fault in Our Stars

"Thomas Edison's last words were "*It's very beautiful over there*". I don't know where there is, but I believe it's somewhere, and I hope it's beautiful."
— **John Green,** Looking for Alaska

THANKYOU!!! BYEE!!!!

DOCKER VOLUMES

In Docker, a volume is a way to store and manage **persistent data** that can be shared among containers.
Therefore, even if we remove a container, still the data inside the container will be retained/ available.

There are 3 types of volumes in Docker:

1. HOST VOLUME:
This type of volume is created by mounting a directory from the host system into the container. The data in the volume is stored on the **host's filesystem**, and can be accessed by the container even after it is stopped and restarted.

2. NAMED VOLUME:
A named volume is created and managed by **Docker**. It can be used to share data between containers, and can be backed up, moved, and restored as a single entity.

3. ANONYMOUS VOLUME:
An anonymous volume is created automatically by Docker when a container is started and a volume is specified in the container's configuration. The volume is given a random name and is not intended to be shared between containers.

Here, we are first learning the **HOST** volume.

Practical : Docker Volume -> Host Volume

1. Creating a directory /data on my host machine. Adding a file in the directory.

```
ubuntu $ mkdir /data
ubuntu $ cd /d
data/ dev/
ubuntu $ cd /data
ubuntu $ touch file1.txt
ubuntu $ ls
file1.txt
ubuntu $ echo "Hello Connections!!"
echo "Hello Connectionsls"
Hello Connectionsls
ubuntu $ echo "Hello Connections" > file1.txt
ubuntu $ ls
file1.txt
ubuntu $ cat file1.txt
Hello Connections
```

2. Running an ubuntu container.
Docker volume => " -v " source: destination

```
ubuntu $ docker run --name=testvol -v /data:/mydata -it ubuntu bash
root@33559538da57:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  mydata  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@33559538da57:/# cd mydata/
root@33559538da57:/mydata# ls
file1.txt
root@33559538da57:/mydata# cat file1.txt
Hello Connections
```

Voila!! The data in my host machine's directory /data is visible in /mydata directory of the container. This is Docker Host Volume.

3. Removing the container. Running a new container with a different image, mounting it with the volume.

```
ubuntu $ docker rm testvol
testvol
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ubuntu $ docker run --name=testvol2 -v /data:/mydata -it docker.io/centos:7 bash
Unable to find image 'centos:7' locally
7: Pulling from library/centos
2d473b07cdd5: Pull complete
Digest: sha256:be65f488b7764ad3638f236b7b515b3678369a5124c47b8d32916d6487418ea4
Status: Downloaded newer image for centos:7
```

4. Checking whether the data is lost or available:

```
[root@cc0c2dffaf32 /]# ls
anaconda-post.log  bin  dev  etc  home  lib  lib64  media  mnt  mydata  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
[root@cc0c2dffaf32 /]# cd /m
media/  mnt/    mydata/
[root@cc0c2dffaf32 /]# cd /mydata/
[root@cc0c2dffaf32 mydata]# ls
file1.txt
[root@cc0c2dffaf32 mydata]# cat file1.txt
Hello Connections
```

This proves that even if we remove a container, still the data inside it remains persistent.

---

1. Pulling the httpd image & inspecting it.

```
ubuntu $ docker pull httpd
Using default tag: latest
latest: Pulling from library/httpd
3f9582a2cbe7: Pull complete
9423d69c3be7: Pull complete
d1f584c02b5d: Pull complete
758a20a64707: Pull complete
08507f82f391: Pull complete
Digest: sha256:76618ddd53f315a1436a56dc84ad57032e1b2123f2f6489ce9c575c4b280c4f4
Status: Downloaded newer image for httpd:latest
docker.io/library/httpd:latest
ubuntu $ docker inspect httpd
```

Here we will find either Working dir / Volume. It has a path where the container's data will be stored. Here Working dir = /usr/local/apache2

```
"Image": "sha256:a5930d7e9b3c7fa530cff2806db329c228e08a75d9db73314b3b5724c0858b60",
"Volumes": null,
"WorkingDir": "/usr/local/apache2",
"Entrypoint": null,
"OnBuild": null,
"Labels": {},
```

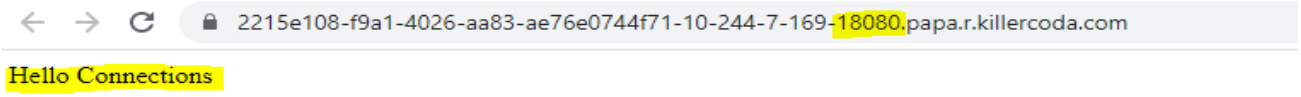
2. Creating a dir on the host machine : /code, and creating an index.html file.

```
ubuntu $ mkdir /code
ubuntu $ cd /code
ubuntu $ echo "Hello Connections" > index.html
ubuntu $ cat index.html
Hello Connections
```

3. Creating a container with httpd image, and attaching volume.

```
ubuntu $ docker run --name=test -v /code:/usr/local/apache2/htdocs -p 18080:80 -d httpd
b8620cd32ae27657e4085438ca3c40242e76e004ec8c8f23f56053c6411263a4
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED         STATUS         PORTS                                      NAMES
b8620cd32ae2   httpd     "httpd-foreground"   5 seconds ago   Up 4 seconds   0.0.0.0:18080->80/tcp, :::18080->80/tcp   test
ubuntu $
```

4. Verifying;



PRACTICAL - DOCKER VOLUMES -> NAMED VOLUMES

Here, the data / directory is managed by **Docker Daemon.**
Allows a 3rd party volume to connect to the container.
Here we don't have to worry about creating the directory manually.

Creating a Volume : **$docker volume create <name>**

```
ubuntu $ docker volume create myvol
myvol
ubuntu $ docker volume ls
DRIVER    VOLUME NAME
local     myvol
ubuntu $ docker volume inspect myvol
[
    {
        "CreatedAt": "2023-03-18T10:18:52Z",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/myvol/_data",
        "Name": "myvol",
        "Options": {},
        "Scope": "local"
    }
]
```

Here we created a Volume -> **"myvol"**.
Listed the Volumes .
Inspected the "myvol" volume.
Here, we get the **Mount Point** i.e. the path where the data will be stored in the volume.

Running an **ubuntu container**, and attaching the created volume "myvol" to it.
Creating some files in /mydata dir of the container which is mapped to the volume.

```
ubuntu $ docker run -v myvol:/mydata -it ubuntu bash
root@af9d51581374:/# ls
bin   dev  home  lib32  libx32  mnt     opt   root  sbin  sys  usr
boot  etc  lib   lib64  media   mydata  proc  run   srv   tmp  var
root@af9d51581374:/# cd mydata
root@af9d51581374:/mydata# touch file{1..100}.png
root@af9d51581374:/mydata# ls
file1.png    file18.png  file27.png  file36.png  file45.png  file54.png  file63.png  file72.png  file81.png  file90.png
file10.png   file19.png  file28.png  file37.png  file46.png  file55.png  file64.png  file73.png  file82.png  file91.png
file100.png  file2.png   file29.png  file38.png  file47.png  file56.png  file65.png  file74.png  file83.png  file92.png
file11.png   file20.png  file3.png   file39.png  file48.png  file57.png  file66.png  file75.png  file84.png  file93.png
file12.png   file21.png  file30.png  file4.png   file49.png  file58.png  file67.png  file76.png  file85.png  file94.png
file13.png   file22.png  file31.png  file40.png  file5.png   file59.png  file68.png  file77.png  file86.png  file95.png
file14.png   file23.png  file32.png  file41.png  file50.png  file6.png   file69.png  file78.png  file87.png  file96.png
file15.png   file24.png  file33.png  file42.png  file51.png  file60.png  file7.png   file79.png  file88.png  file97.png
file16.png   file25.png  file34.png  file43.png  file52.png  file61.png  file70.png  file8.png   file89.png  file98.png
file17.png   file26.png  file35.png  file44.png  file53.png  file62.png  file71.png  file80.png  file9.png   file99.png
root@af9d51581374:/mydata# cd ..
root@af9d51581374:/# exit
exit
```

Checking whether these files are available or not in the volume:

```
ubuntu $ docker volume inspect myvol
[
    {
        "CreatedAt": "2023-03-18T10:23:00Z",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/myvol/_data",
        "Name": "myvol",
        "Options": {},
        "Scope": "local"
    }
]
ubuntu $ cd /var/lib/docker/volumes/myvol/_data
ubuntu $ ls
file1.png    file18.png  file27.png  file36.png  file45.png  file54.png  file63.png  file72.png  file81.png  file90.png
file10.png   file19.png  file28.png  file37.png  file46.png  file55.png  file64.png  file73.png  file82.png  file91.png
file100.png  file2.png   file29.png  file38.png  file47.png  file56.png  file65.png  file74.png  file83.png  file92.png
file11.png   file20.png  file3.png   file39.png  file48.png  file57.png  file66.png  file75.png  file84.png  file93.png
file12.png   file21.png  file30.png  file4.png   file49.png  file58.png  file67.png  file76.png  file85.png  file94.png
file13.png   file22.png  file31.png  file40.png  file5.png   file59.png  file68.png  file77.png  file86.png  file95.png
file14.png   file23.png  file32.png  file41.png  file50.png  file6.png   file69.png  file78.png  file87.png  file96.png
file15.png   file24.png  file33.png  file42.png  file51.png  file60.png  file7.png   file79.png  file88.png  file97.png
```

And Yes!! Voila, They all are available.

PRACTICAL - DOCKER VOLUMES -> MANAGED VOLUMES

(Note: We call it Managed Volumes)

#Storing the data in custom directory

Creating a volume **“vol2”** using parameters.
Device = our customised path => data will be stored in this dir on the volume.

```
ubuntu $ docker volume create --driver local --opt type=none --opt device=/kd --opt o=bind vol2
vol2
ubuntu $ docker volume ls
DRIVER    VOLUME NAME
local     myvol
local     vol2
```

Creating the dir /kd, and inspecting the volume

```
ubuntu $ mkdir /kd
mkdir: cannot create directory '/kd': File exists
ubuntu $ docker volume inspect vol2
[
    {
        "CreatedAt": "2023-03-18T10:35:27Z",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/vol2/_data",
        "Name": "vol2",
        "Options": {
            "device": "/kd",
            "o": "bind",
            "type": "none"
        },
        "Scope": "local"
    }
]
```

Running an **Ubuntu container** and **attaching vol2 with the container.**
Creating some data inside the container.
Rather than viewing the data in the Mount Path of volume, we can view the data in our dir /kd on the host machine, which is now our customised path for storing data in volumes.

```
ubuntu $ docker run -v vol2:/kd -it ubuntu bash
root@a733ed3d56dd:/# ls
bin   dev  home  lib    lib64   media  opt   root  sbin  sys  usr
boot  etc  kd    lib32  libx32  mnt    proc  run   srv   tmp  var
root@a733ed3d56dd:/# cd /kd
root@a733ed3d56dd:/kd# touch file{1..4}.txt
root@a733ed3d56dd:/kd# ls
file1.txt  file2.txt  file3.txt  file4.txt
root@a733ed3d56dd:/kd# cd ..
root@a733ed3d56dd:/# exit
exit
ubuntu $ ls /
bin   dev  home  lib    lib64   lost+found  mnt  proc  run   snap  swapfile  tmp  var
boot  etc  kd    lib32  libx32  media       opt  root  sbin  srv   sys       usr
ubuntu $ cd /kd
ubuntu $ ls
file1.txt  file2.txt  file3.txt  file4.txt
ubuntu $
```

Task:

Create a custom subnet for a user created network

=>

Creating a Network “mynet1”, and listing the networks.

```
ubuntu $ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
db8bda17d0c7   bridge    bridge    local
a4ed26aa0e2d   host      host      local
790fa6da653d   none      null      local
ubuntu $ docker network create mynet1
6fb1284d3d264a6838e12783d78edc8478a105be2c8e7ebd29e97d15b2a2386a
ubuntu $ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
db8bda17d0c7   bridge    bridge    local
a4ed26aa0e2d   host      host      local
6fb1284d3d26   mynet1    bridge    local
790fa6da653d   none      null      local
ubuntu $
```

Inspecting the “mynet1” network:

```
{
    "Subnet": "172.18.0.0/16",
    "Gateway": "172.18.0.1"
}
```

We need to customise this subnet.

```
Create a network
ubuntu $ docker network create --subnet 192.168.0.0 mynet2
Error response from daemon: Invalid subnet 192.168.0.0 : invalid CIDR address: 192.168.0.0
ubuntu $ docker network create --subnet 192.168.0.0/16 mynet2
06aa6a43460cbb6741aba5f35b2de288c486635d43a6ac34d706ed9a906d3a00
ubuntu $ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
db8bda17d0c7   bridge    bridge    local
a4ed26aa0e2d   host      host      local
6fb1284d3d26   mynet1    bridge    local
06aa6a43460c   mynet2    bridge    local
790fa6da653d   none      null      local
```

Inspecting the network “mynet2” and checking:

```
ubuntu $ docker network inspect mynet2
[
    {
        "Name": "mynet2",
        "Id": "06aa6a43460cbb6741aba5f35b2de288c486635d43a6ac34d706ed9a906d3a00",
        "Created": "2023-03-18T10:56:22.24628373Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "192.168.0.0/16"
                }
            ]
```

Yess!! The subnet of “mynet2” network has been customised.

History of commands:

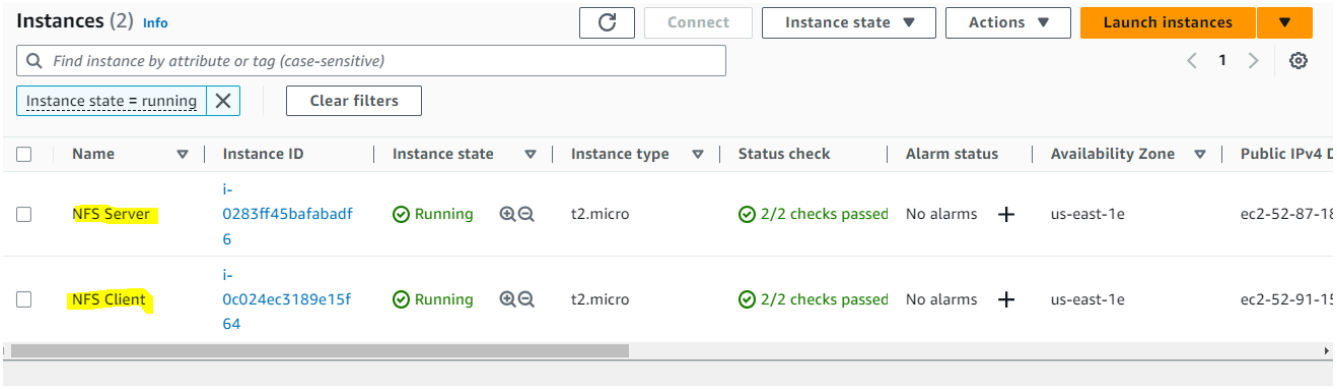
```
 32  docker network ls
 33  docker network create mynet1
 34  docker network ls
 35  docker network inspect mynet1
 36  docker network --help
 37  docker network create --help
 38  docker network create --subnet --help
 39  docker network create --subnet 192.168.0.0 mynet2
 40  docker network create --subnet 192.168.0.0/16 mynet2
 41  docker network ls
 42  docker network inspect mynet2
 43  history
```

Lets try customising the **Gateway** too:

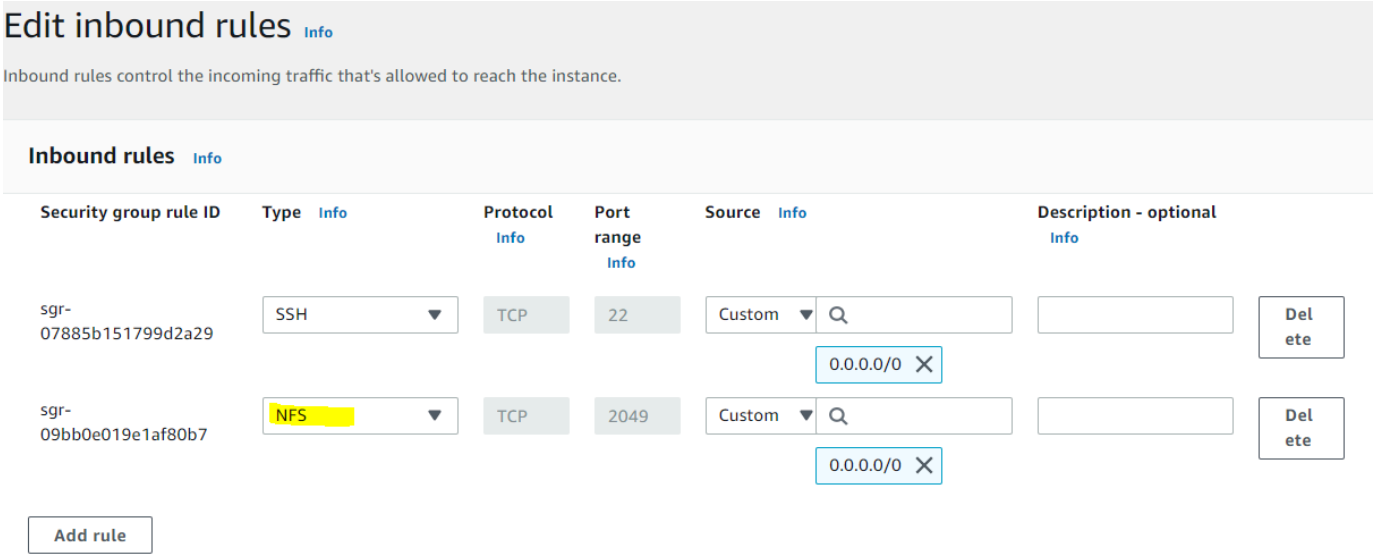
```
ubuntu $ docker network create --subnet 192.167.0.0/16 --gateway 192.167.10.1 mynet3
1e50ca7a09a2fd260c40b4206b11a651f7cd4dbe409776aef3f09684092c08fc
ubuntu $ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
db8bda17d0c7   bridge    bridge    local
a4ed26aa0e2d   host      host      local
6fb1284d3d26   mynet1    bridge    local
06aa6a43460c   mynet2    bridge    local
1e50ca7a09a2   mynet3    bridge    local
790fa6da653d   none      null      local
ubuntu $ docker network inspect mynet3
[
    {
        "Name": "mynet3",
        "Id": "1e50ca7a09a2fd260c40b4206b11a651f7cd4dbe409776aef3f09684092c08fc",
        "Created": "2023-03-18T11:00:41.17148202Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "192.167.0.0/16",
                    "Gateway": "192.167.10.1"
```

NFS SERVER USING DOCKER VOLUME: [paused]

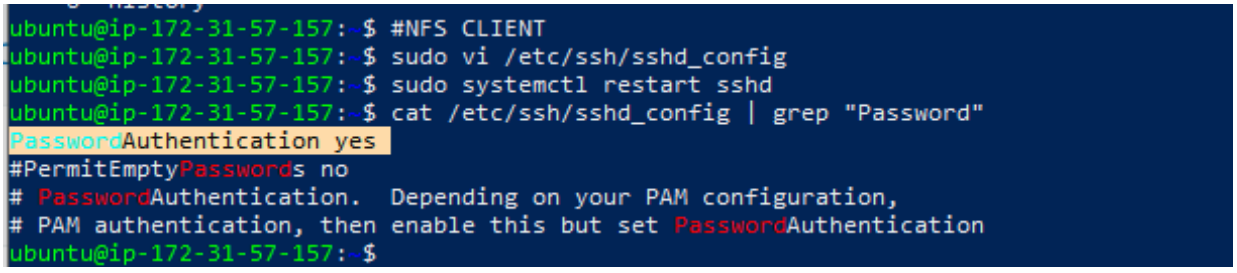
Creating 2 Ubuntu Instances: NFS Server & NFS Client
Instance1 = NFS Server
Instance2 = NFS Client



Add NFS in the Inbound rules of Security group:



Take ssh access to Instance 2 => NFS Client, and make Password authentication =”yes”.



Change the Passwd of user “ubuntu”

```
ubuntu@ip-172-31-57-157:~$ sudo passwd ubuntu
New password:
Retype new password:
passwd: password updated successfully
ubuntu@ip-172-31-57-157:~$
```

Taking ssh access to Instance1 => NFS Server:

```
ubuntu@ip-172-31-62-93:~$ history
    1  #NFS SERVER
    2  sudo apt install nfs-kernel-server
    3  sudo mkdir /mydata
    4  sudo vi /etc/exports
    5  sudo exportfs -r
    6  sudo chmod 777 /mydata
    7  sudo chmod 777 /mydata/
    8  sudo systemctl restart nfs-server
    9  sudo systemctl enable nfs-server
   10  history
ubuntu@ip-172-31-62-93:~$ cat /etc/exports
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)

/mydata *(rw,sync,no_root_squash)
ubuntu@ip-172-31-62-93:~$
```

On Instance1, take access of Instance2 => NFS CLient using its Private IP:
$**ssh ubuntu@<private_ip_of_instance_2>**

```
   17  #NFS CLIENT
   18  docker
   19  sudo apt update
   20  sudo apt install docker.io
   21  docker --help
   22  sudo apt install nfs-common
   23  sudo mount -t nfs 172.31.62.93:/mydata /mnt
   24  df -h
   25  history
ubuntu@ip-172-31-57-157:~$ 24!
24!: command not found
ubuntu@ip-172-31-57-157:~$ !24
df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/root             7.6G  2.0G  5.6G  27% /
tmpfs                 484M     0  484M   0% /dev/shm
tmpfs                 194M  868K  193M   1% /run
tmpfs                 5.0M     0  5.0M   0% /run/lock
/dev/xvda15           105M  6.1M   99M   6% /boot/efi
tmpfs                  97M  4.0K   97M   1% /run/user/1000
172.31.62.93:/mydata  7.6G  1.6G  6.1G  21% /mnt
ubuntu@ip-172-31-57-157:~$
```

Verifying by mounting the directory. =>$sudo umount /mnt

Creating a volume “myvol” (managed volume)

```
ubuntu@ip-172-31-57-157:~$ sudo docker volume create --driver local --opt type=nfs --opt o=addr=172.31.62.93,rw --opt device=/mydata myvol
myvol
ubuntu@ip-172-31-57-157:~$ sudo docker volume ls
DRIVER    VOLUME NAME
local     myvol
ubuntu@ip-172-31-57-157:~$ sudo docker volume inspect myvol
[
    {
        "CreatedAt": "2023-03-21T07:16:05Z",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/myvol/_data",
        "Name": "myvol",
        "Options": {
            "device": "/mydata",
            "o": "addr=172.31.62.93,rw",
            "type": "nfs"
        },
        "Scope": "local"
    }
]
```

CAPACITY PLANNING
[CAPPING THE MEMORY]

Capacity planning in Docker involves determining the resources required to run Docker containers and ensuring that the available resources are sufficient to meet the needs of the containers that are running.

$docker run - - help:

```
ubuntu $ docker run --help | grep memory
      --kernel-memory bytes            Kernel memory limit
  -m, --memory bytes                   Memory limit
      --memory-reservation bytes       Memory soft limit
      --memory-swap bytes              Swap limit equal to memory plus swap: '-1' to enable unlimited swap
      --memory-swappiness int          Tune container memory swappiness (0 to 100) (default -1)
```

Memory can be limited in 2 ways:

1. Maximum Limit => memory-bytes

   This limits the memory usage upto a maximum limit that we specify.
   Can be specified using the parameter “-m”
   $docker run -m <limit> -d <docker_image>

2. Minimum Limit => memory-reservation bytes

   This limits the memory usage from a minimum limit that we specify.
   Can be specified using the parameter “--memory-reservation”
   $docker run - - memory-reservation=<limit> -d <docker_image>

Pulled a httpd image, and created 2 containers : demo1 and demo2:

```
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED      SIZE
httpd        latest    192d41583429   4 days ago   145MB
ubuntu $ docker run --name=demo1 -d docker.io/httpd
55f7462747a0f031016ad350b51790616c69c083a3a7712d69cff3a4d6ea13eb
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED         STATUS        PORTS     NAMES
55f7462747a0   httpd     "httpd-foreground"   2 seconds ago   Up 1 second   80/tcp    demo1
ubuntu $ docker run --name=demo2 -d docker.io/httpd
9bfba7233ba3d7e95ebcd8ca8db5f114f236e1c589a26619c944c3f252a493fc
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED          STATUS          PORTS     NAMES
9bfba7233ba3   httpd     "httpd-foreground"   2 seconds ago    Up 2 seconds    80/tcp    demo2
55f7462747a0   httpd     "httpd-foreground"   18 seconds ago   Up 17 seconds   80/tcp    demo1
```

$docker stats
(It will show all the statistics, use of resources by the containers)

```
Editor   Tab 1   +
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT     MEM %     NET I/O       BLOCK I/O     PIDS
9bfba7233ba3   demo2     0.00%     6.203MiB / 1.937GiB   0.31%     15kB / 0B     127kB / 0B    82
55f7462747a0   demo1     0.00%     6.098MiB / 1.937GiB   0.31%     15.9kB / 0B   0B / 0B       82
```

Creating a docker container => with maximum memory usage limit as 700MiB

```
ubuntu $ docker run --name test1 -m 700m -d docker.io/httpd
WARNING: Your kernel does not support swap limit capabilities or the cgroup is not mounted. Memory limited without swap.
13a0a12d045d36de3df5bfa3972ec033671b955cae0e007ec772292c2bea45c9
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED          STATUS          PORTS     NAMES
13a0a12d045d   httpd     "httpd-foreground"   4 seconds ago    Up 2 seconds    80/tcp    test1
9bfba7233ba3   httpd     "httpd-foreground"   22 minutes ago   Up 22 minutes   80/tcp    demo2
55f7462747a0   httpd     "httpd-foreground"   22 minutes ago   Up 22 minutes   80/tcp    demo1
```

Let's view the statistics:

$docker stats

```
Editor   Tab 1   +
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT     MEM %     NET I/O       BLOCK I/O     PIDS
13a0a12d045d   test1     0.00%     6.074MiB / 700MiB     0.87%     4.22kB / 0B   0B / 0B       82
9bfba7233ba3   demo2     0.00%     6.203MiB / 1.937GiB   0.31%     56.8kB / 0B   127kB / 0B    82
55f7462747a0   demo1     0.00%     6.098MiB / 1.937GiB   0.31%     57.7kB / 0B   0B / 0B       82
```

Container test1 shows the maximum Memory usage limit as 700MiB.
Therefore, the container won't consume more than 700MiB of memory.

Now, Let's try creating a container with Minimum memory usage limit:

```
ubuntu $ docker run --name test2 --memory-reservation 700m -d docker.io/httpd
c156ee8e381cba5e168ed6528b4d6888baca04edab26fb50fa50ec14d82ff004
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED          STATUS          PORTS     NAMES
c156ee8e381c   httpd     "httpd-foreground"   2 seconds ago    Up 2 seconds    80/tcp    test2
13a0a12d045d   httpd     "httpd-foreground"   4 minutes ago    Up 4 minutes    80/tcp    test1
9bfba7233ba3   httpd     "httpd-foreground"   26 minutes ago   Up 26 minutes   80/tcp    demo2
55f7462747a0   httpd     "httpd-foreground"   26 minutes ago   Up 26 minutes   80/tcp    demo1
```

Created a container "test2" with minimum memory reservation = 700 MiB, this means that the system must have 700 MiB memory free. Only then will the container get created.

$docker stats

```
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT     MEM %     NET I/O       BLOCK I/O     PIDS
c156ee8e381c   test2     0.01%     6.254MiB / 1.937GiB   0.32%     3.19kB / 0B   483kB / 0B    82
13a0a12d045d   test1     0.00%     6.074MiB / 700MiB     0.87%     16.7kB / 0B   0B / 0B       82
9bfba7233ba3   demo2     0.00%     6.203MiB / 1.937GiB   0.31%     69.1kB / 0B   127kB / 0B    82
55f7462747a0   demo1     0.00%     6.098MiB / 1.937GiB   0.31%     70kB / 0B     0B / 0B       82
```

```
ubuntu $ docker inspect test2 | grep Memory
            "Memory": 0,
            "KernelMemory": 0,
            "KernelMemoryTCP": 0,
            "MemoryReservation": 734003200,
            "MemorySwap": 0,
            "MemorySwappiness": null,
```

CAPPING THE CPU SHARES

We can limit the amount of CPU usage by limiting the CPU Shares.
This can be done by using the parameter “-c” or “- - cpu-shares” followed by the number of cpu.
$docker run --help

```
-c, --cpu-shares int                   CPU shares (relative weight)
    --cpus decimal                     Number of CPUs
    --cpuset-cpus string               CPUs in which to allow execution (0-3, 0,1)
    --cpuset-mems string               MEMs in which to allow execution (0-3, 0,1)
```

$lscpu
This will show the details about the cpu.
Note: 1 vcpu = 1000 mcpu’s

```
ubuntu $ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   46 bits physical, 48 bits virtual
CPU(s):                          1
```

Created a container “test” using docker.io/httpd image.
Using the “-c” parameter, specifying the CPU shares => 200 i.e. (200 mcpu’s out of 1000)

```
ubuntu $ docker run --name test -c "200" -d docker.io/httpd
abb4a02f51394a51b2cbdd9481c099944a6b2ead5a5e4a8686fe9350debdd875
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED         STATUS         PORTS     NAMES
abb4a02f5139   httpd     "httpd-foreground"   3 seconds ago   Up 2 seconds   80/tcp    test
```

Inspecting the docker container “test’

```
ubuntu $ docker inspect test | grep Cpu
            "CpuShares": 200,
            "NanoCpus": 0,
            "CpuPeriod": 0,
            "CpuQuota": 0,
            "CpuRealtimePeriod": 0,
            "CpuRealtimeRuntime": 0,
            "CpusetCpus": "",
            "CpusetMems": "",
            "CpuCount": 0,
            "CpuPercent": 0,
ubuntu $
```

VIRTUAL ETHERNET (Veth)

In Docker, a "veth" is a virtual network interface device that is used to connect Docker containers to each other and to the host system.
Whenever we create a Docker network, a virtual bridge / veth is created on the host system, which is used to connect containers to each other and to the outside world.

Created 2 containers from docker.io/httpd image

```
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED      SIZE
httpd        latest    192d41583429   5 days ago   145MB
ubuntu $ docker run --name=cont1 -d docker.io/httpd
25cb3488c0d040aabce7205ff9252191fe1e6f3e384a238db9d7c045e3743dbd
^[[Aubuntu $
ubuntu $ docker run --name=cont2 -d docker.io/httpd
acab0734388ca087236fc5637524badf8d2d6b8461972437bcda3acad47e25f7
ubuntu $ docker container ls
CONTAINER ID   IMAGE     COMMAND              CREATED          STATUS          PORTS     NAMES
acab0734388c   httpd     "httpd-foreground"   8 seconds ago    Up 7 seconds    80/tcp    cont2
25cb3488c0d0   httpd     "httpd-foreground"   25 seconds ago   Up 24 seconds   80/tcp    cont1
```

Let's check the number of veth's created on our host machine.
$ip a
Or
$ifconfig

```
ubuntu $ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc fq_codel state UP group default qlen 1000
    link/ether f2:05:f6:3f:86:80 brd ff:ff:ff:ff:ff:ff
    inet 172.30.1.2/24 brd 172.30.1.255 scope global dynamic enp1s0
       valid_lft 86312390sec preferred_lft 86312390sec
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1454 qdisc noqueue state UP group default
    link/ether 02:42:fb:12:3b:47 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:fbff:fe12:3b47/64 scope link
       valid_lft forever preferred_lft forever
21: veth394849e@if20: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1454 qdisc noqueue master docker0 state UP group default
    link/ether de:8b:45:4a:b3:dc brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::dc8b:45ff:fe4a:b3dc/64 scope link
       valid_lft forever preferred_lft forever
23: veth32d1796@if22: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1454 qdisc noqueue master docker0 state UP group default
    link/ether 4e:2b:a4:9d:19:1e brd ff:ff:ff:ff:ff:ff link-netnsid 1
    inet6 fe80::4c2b:a4ff:fe9d:191e/64 scope link
```

Thus, here we have 2 veth's on our host machine. This is because we created 2 containers.
But, how to identify which veth is for which container??

Going inside my "cont1" using exec.

/sys/devices/virtual/net/eth0 is the directory which stores some details, including the **ifindex.**

Here, ifindex is 20.

```
ubuntu $ docker exec -it cont1 bash
root@48904a3c9e83:/usr/local/apache2# find / -name ifindex
find: '/proc/7/map_files': Permission denied
find: '/proc/8/map_files': Permission denied
find: '/proc/9/map_files': Permission denied
/sys/devices/virtual/net/eth0/ifindex
/sys/devices/virtual/net/lo/ifindex
root@48904a3c9e83:/usr/local/apache2# cd /sys/devices/virtual/net/eth0/
root@48904a3c9e83:/sys/devices/virtual/net/eth0# ls
addr_assign_type  carrier_changes     dormant           ifindex          netdev_group  power       subsystem
addr_len          carrier_down_count  duplex            iflink           operstate     proto_down  tx_queue_len
address           carrier_up_count    flags             link_mode        phys_port_id  queues      type
broadcast         dev_id              gro_flush_timeout mtu              phys_port_name speed      uevent
carrier           dev_port            ifalias           name_assign_type phys_switch_id statistics
root@48904a3c9e83:/sys/devices/virtual/net/eth0# cat ifindex
20
```

$ip a

Here, the first veth has **if20** written at the end. Thus, this veth is connected to my First container : **“cont1”**

```
ubuntu $ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc fq_codel state UP group default qlen 1000
    link/ether f2:05:f6:3f:86:80 brd ff:ff:ff:ff:ff:ff
    inet 172.30.1.2/24 brd 172.30.1.255 scope global dynamic enp1s0
       valid_lft 86311821sec preferred_lft 86311821sec
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1454 qdisc noqueue state UP group default
    link/ether 02:42:fb:12:3b:47 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:fbff:fe12:3b47/64 scope link
       valid_lft forever preferred_lft forever
21: veth394849e@if20: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1454 qdisc noqueue master docker0 state UP group default
    link/ether de:8b:45:4a:b3:dc brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::dc8b:45ff:fe4a:b3dc/64 scope link
       valid_lft forever preferred_lft forever
23: veth32d1796@if22: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1454 qdisc noqueue master docker0 state UP group default
    link/ether 4e:2b:a4:9d:19:1e brd ff:ff:ff:ff:ff:ff link-netnsid 1
    inet6 fe80::4c2b:a4ff:fe9d:191e/64 scope link
       valid_lft forever preferred_lft forever
```

And therefore, the 2nd veth is connected to “cont2”

```
ubuntu $ docker exec -it cont2 bash
root@d2d3ae9f2d09:/usr/local/apache2# cd /sys/devices/virtual/net/eth0
root@d2d3ae9f2d09:/sys/devices/virtual/net/eth0# ls
addr_assign_type  carrier_changes     dormant           ifindex          netdev_group    power       subsystem
addr_len          carrier_down_count  duplex            iflink           operstate       proto_down  tx_queue_len
address           carrier_up_count    flags             link_mode        phys_port_id    queues      type
broadcast         dev_id              gro_flush_timeout mtu              phys_port_name  speed       uevent
carrier           dev_port            ifalias           name_assign_type phys_switch_id  statistics
root@d2d3ae9f2d09:/sys/devices/virtual/net/eth0# cat ifindex
22
root@d2d3ae9f2d09:/sys/devices/virtual/net/eth0#
```

RESTRICTING THE ACCESS TO THE CONTAINER

Creating a container “demo1” using docker.io/httpd image.

```
ubuntu $ docker run --name=demo1 -p 18080:80 -d docker.io/httpd
Unable to find image 'httpd:latest' locally
latest: Pulling from library/httpd
f1f26f570256: Pull complete
a6b093ae1967: Pull complete
6b400bbb27df: Pull complete
d9833ead928a: Pull complete
ace056404ed3: Pull complete
Digest: sha256:f3e9eb9acace5bbc13c924293d2247a65bb59b8f062bcd98cd87ee4e18f86733
Status: Downloaded newer image for httpd:latest
cfc4e603d2ab1a16dfe6c4af7ec39403bdcac896afebf5dd0bf4820686e5c6fa
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED         STATUS         PORTS
          NAMES
cfc4e603d2ab   httpd     "httpd-foreground"   5 seconds ago   Up 3 seconds   0.0.0.0:18080->80/tcp, :::18080-
>80/tcp   demo1
ubuntu $
```

Let’s try curling the localhost, 127.0.0.1 (used for testing purpose), ip address of host, secondary ip address of host.

All of them will show the output as **0.0.0.0/18080->80, i.e. can be accessed from “Anywhere”.**

```
ubuntu $ curl localhost:18080
<html><body><h1>It works!</h1></body></html>
ubuntu $ curl 127.0.0.1:18080
<html><body><h1>It works!</h1></body></html>
ubuntu $ curl 172.30.1.2:18080
<html><body><h1>It works!</h1></body></html>
ubuntu $ curl 172.17.0.1:18080
<html><body><h1>It works!</h1></body></html>
ubuntu $
```

But if we want only a certain machine to access our container, then:

$docker run –name <name> -p <ip>:ports -d <docker_image>

Where, ip => the ip which should be able to access the container.

Here, ip = ip address of host machine

```
ubuntu $ docker run --name=demo2 -p 172.30.1.2:28080:80 -d docker.io/httpd
8fc8d0b4fa1dfc39d99de999539114f441a67ec314ef5e507671c328881b4705
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED          STATUS          PORTS
           NAMES
8fc8d0b4fa1d   httpd     "httpd-foreground"   5 seconds ago    Up 4 seconds    172.30.1.2:28080->80/tcp
           demo2
cfc4e603d2ab   httpd     "httpd-foreground"   15 minutes ago   Up 15 minutes   0.0.0.0:18080->80/tcp, :::1808
0->80/tcp   demo1
ubuntu $ curl 172.30.1.2:28080
<html><body><h1>It works!</h1></body></html>
ubuntu $ curl localhost:28080
curl: (7) Failed to connect to localhost port 28080: Connection refused
ubuntu $ curl 127.0.0.1:28080
curl: (7) Failed to connect to 127.0.0.1 port 28080: Connection refused
```

Here, as we have only allowed the host to access the container, thus all others were given an error while curling.

CREATING 2 INTERFACES (NETWORK) ON A SINGLE CONTAINER

Creating a container “demo” using docker.io/busybox image.
Getting inside the container using exec.
“ip a” will show the details, where we can see 1 interface “eth0”.

```
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
busybox      latest    7cfbbec8963d   12 days ago   4.86MB
ubuntu $ docker run -itd --name demo docker.io/busybox
786668f02e18bb9aaf4ca5934da1791be8c0a6b25bd404660e9e863880c49376
ubuntu $ docker container ls
CONTAINER ID   IMAGE     COMMAND   CREATED          STATUS          PORTS     NAMES
786668f02e18   busybox   "sh"      4 seconds ago    Up 4 seconds              demo
ubuntu $ docker exec -it demo sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
10: eth0@if11: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1454 qdisc noqueue
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
```

Whenever we create a container, by default “bridge” is the network.

```
  "NetworkSettings": {
      "Bridge": "",
      "SandboxID": "2e6c4530494303cceef6c49a4fd340cb0579ba51e45de957690153a255a6fbb0",
      "HairpinMode": false,
      "LinkLocalIPv6Address": "",
      "LinkLocalIPv6PrefixLen": 0,
      "Ports": {}
```

Let’s try connecting the container to another network / interface.
Creating a network “mynet1”

```
ubuntu $ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
db8bda17d0c7   bridge    bridge    local
a4ed26aa0e2d   host      host      local
790fa6da653d   none      null      local
ubuntu $ docker network create mynet1
b653ab9a742d4343808b04e3d2fad978866b1308313788ebcc739bb1b5762ff1
ubuntu $ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
db8bda17d0c7   bridge    bridge    local
a4ed26aa0e2d   host      host      local
b653ab9a742d   mynet1    bridge    local
790fa6da653d   none      null      local
```

Attaching the network “mynet1” to our container “demo”

$docker network connect <network_name> <container_name>

```
ubuntu $ docker container ls
CONTAINER ID   IMAGE     COMMAND   CREATED         STATUS         PORTS     NAMES
786668f02e18   busybox   "sh"      7 minutes ago   Up 7 minutes             demo
ubuntu $ docker network connect mynet1 demo
```

$docker inspect demo


```
"Networks": {
    "bridge": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": null,
        "NetworkID": "db8bda17d0c7f051e6895bf43ab457989fd28abd9d52d46ebc9c018397d18957",
        "EndpointID": "8075fc895728aff8d8f9310d3fb65903b7d51acf316499dde06ecb5452db7a2b",
        "Gateway": "172.17.0.1",
        "IPAddress": "172.17.0.2",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "02:42:ac:11:00:02",
        "DriverOpts": null
    },
    "mynet1": {
        "IPAMConfig": {},
        "Links": null,
        "Aliases": [
            "786668f02e18"
        ],
```


Getting inside the container “demo”

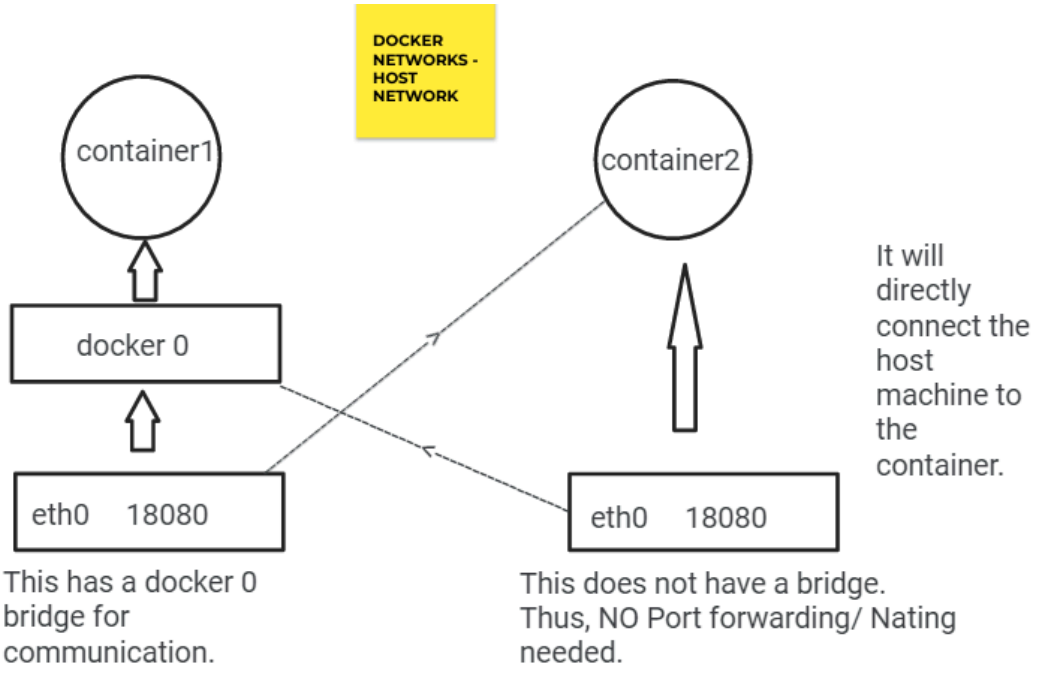
```
ubuntu $ docker exec -it demo sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
10: eth0@if11: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1454 qdisc noqueue
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
13: eth1@if14: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 02:42:ac:12:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.2/16 brd 172.18.255.255 scope global eth1
       valid_lft forever preferred_lft forever
```

Here we can see 2 eth = eth0 and eth1. Thus 2 interfaces / networks have been added to a container.

DOCKER NETWORK - HOST NETWORK

Earlier, we saw the first network type : “Bridge”, which was the default network.
2nd Network type : **HOST NETWORK.**

Host Network doesn’t have any intermediary bridge “docker 0” for communication between the container and the host machine.



Using the host network mode can be useful in certain situations where we need to give a container direct access to the host's network interfaces.
However, using the host network mode also has some drawbacks. One of the main concerns is that it can reduce the level of isolation between the container and the host machine, which can potentially increase the security risk of the container.

Creating a container “test1” using docker.io/httpd image.
Here, we are mentioning the “network” as “host”.
When we curl the localhost, it will show the output.

```
ubuntu $ docker images
REPOSITORY                  TAG       IMAGE ID       CREATED       SIZE
httpd                       latest    192d41583429   5 days ago    145MB
openshift/hello-openshift   latest    7af3297a3fb4   4 years ago   6.09MB
ubuntu $ docker run --name test1 --network host -d docker.io/httpd
53b72d75e91db2ac94cc48ed0250e747293881fa7aff975c4dcb65631901ffe6
ubuntu $ docker container ls
CONTAINER ID   IMAGE     COMMAND              CREATED         STATUS         PORTS     NAMES
53b72d75e91d   httpd     "httpd-foreground"   4 seconds ago   Up 4 seconds             test1
ubuntu $ curl localhost:80
<html><body><h1>It works!</h1></body></html>
ubuntu $ curl localhost
<html><body><h1>It works!</h1></body></html>
```

Now, as port 80 is connected to container “test1”, therefore it is busy for other containers i.e. other containers can’t access port 80. Thus, any container of httpd image won’t be created, as no listening sockets will be available :80.

```
ubuntu $ docker run --name test2 --network host -d docker.io/httpd
a23399424a27b3a3ad89b37c254da7eac5fb3642d1226456692c7a6f912b4ea2
ubuntu $ docker container ls
CONTAINER ID   IMAGE     COMMAND              CREATED         STATUS         PORTS     NAMES
53b72d75e91d   httpd     "httpd-foreground"   3 minutes ago   Up 3 minutes             test1
ubuntu $ docker logs test2
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 127.0.0.1. Set the 'ServerN
ame' directive globally to suppress this message
(98)Address already in use: AH00072: make_sock: could not bind to address [::]:80
(98)Address already in use: AH00072: make_sock: could not bind to address 0.0.0.0:80
no listening sockets available, shutting down
AH00015: Unable to open logs
```

Creating another container, with docker.io/openshift/hello-openshift image.
Now, this won’t give an error, rather will show the output, as the default port of openshift is 8080, which is available.

```
ubuntu $ docker run --name test3 --network host -d docker.io/openshift/hello-openshift
5f7b446e11b2317aca04ca6661b3867fcf3ab30d109a3b0fa3807493bc6b69df
ubuntu $ docker container ls
CONTAINER ID   IMAGE                       COMMAND              CREATED          STATUS          PORTS     NAMES
5f7b446e11b2   openshift/hello-openshift   "/hello-openshift"   5 seconds ago    Up 5 seconds              test3
53b72d75e91d   httpd                       "httpd-foreground"   7 minutes ago    Up 7 minutes              test1
ubuntu $ curl localhost:8080
Hello OpenShift!
```

This command tells us which port is in use / busy.

$ss -tunlp

```
ubuntu $ ss -tunlp | grep 80
tcp    LISTEN  0       4096                    *:8080                *:*      users:(("hello-opens
hift",pid=26968,fd=5))
tcp    LISTEN  0       511                     *:80                  *:*      users:(("httpd",pid=
25988,fd=4),("httpd",pid=25987,fd=4),("httpd",pid=25986,fd=4),("httpd",pid=25966,fd=4))
```

Or

$ss -netstat

```
ubuntu $ ss -netstat | grep 80
LISTEN    0         4096                  *:8080                *:*        uid:1001 ino:88572 s
k:c v6only:0 <->
LISTEN    0         511                   *:80                  *:*        ino:85111 sk:d v6onl
y:0 <->
```

DOCKER NETWORK - NONE NETWORK

This is a completely isolated process/ container.
It has no IP address, no MAC address.
Used for testing purposes

To create a container with network as “none”:

$docker run - - network none <docker_image>

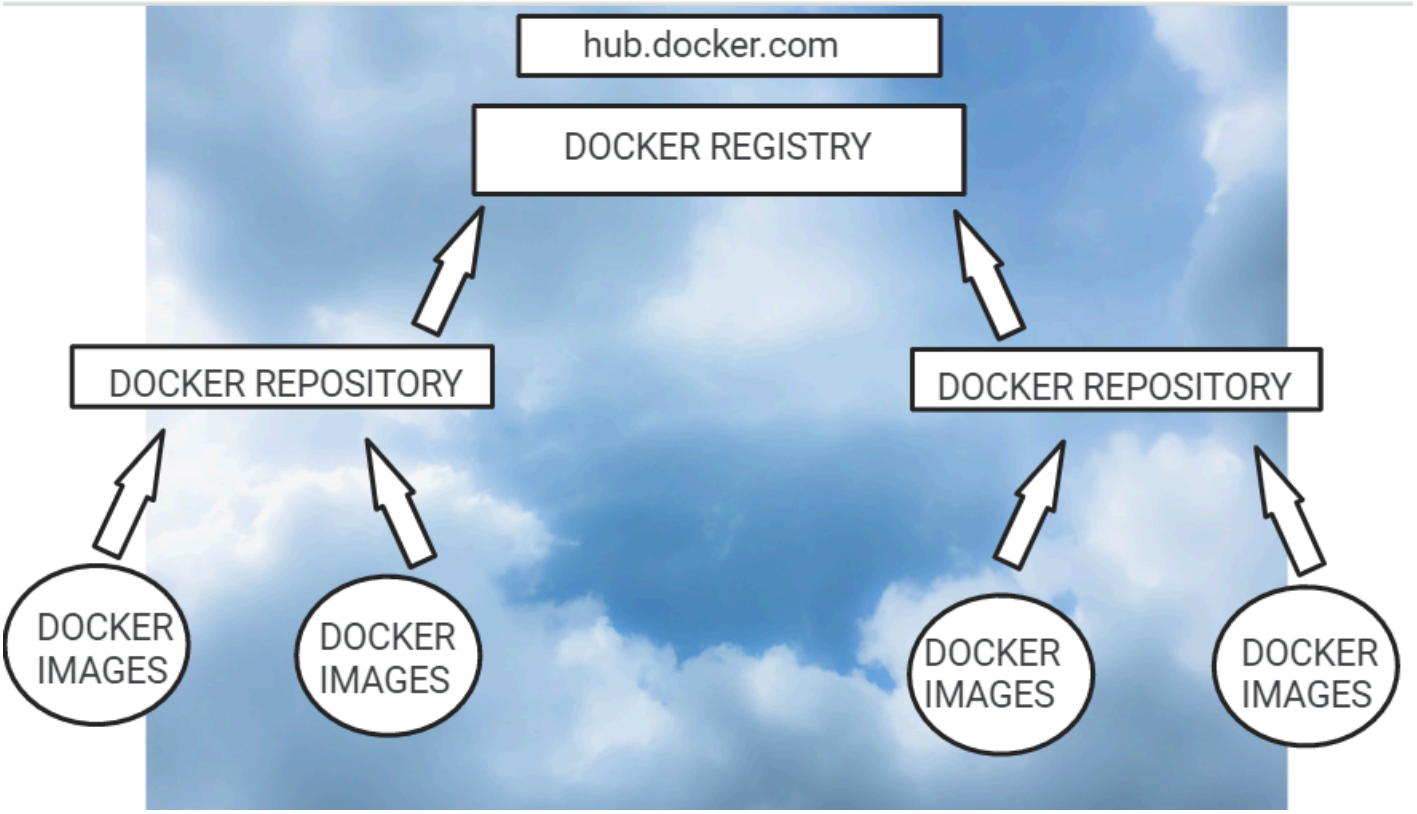
```
ubuntu $ docker run -d --name demo --network none docker.io/httpd
dd6c068f0af6738f14a1255059a23c866f8a6585a30f11edc071d2f172ddd929
ubuntu $ docker container ls
CONTAINER ID   IMAGE     COMMAND              CREATED         STATUS         PORTS     NAMES
dd6c068f0af6   httpd     "httpd-foreground"   3 seconds ago   Up 2 seconds             demo
ubuntu $ docker inspect demo
```

Here, we can see that the container has no IP as well as MAC address

```
    "IPAddress": "",
    "IPPrefixLen": 0,
    "IPv6Gateway": "",
    "MacAddress": "",
    "Networks": {
        "none": {
            "IPAMConfig": null,
            "Links": null,
            "Aliases": null,
```

DOCKER PUSH

The docker push command is used to upload a Docker image to a container registry, such as : docker.io , quay.io , registry.redhat.com, centos.registry.com.

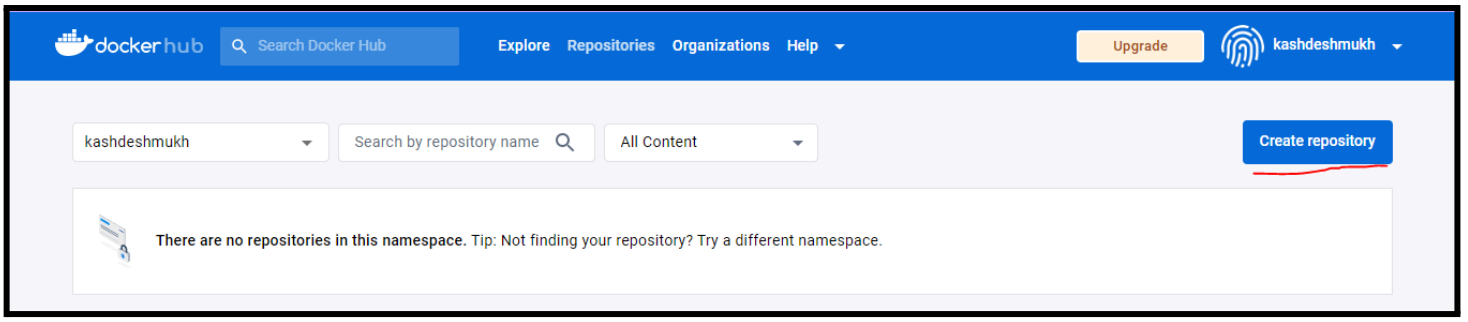


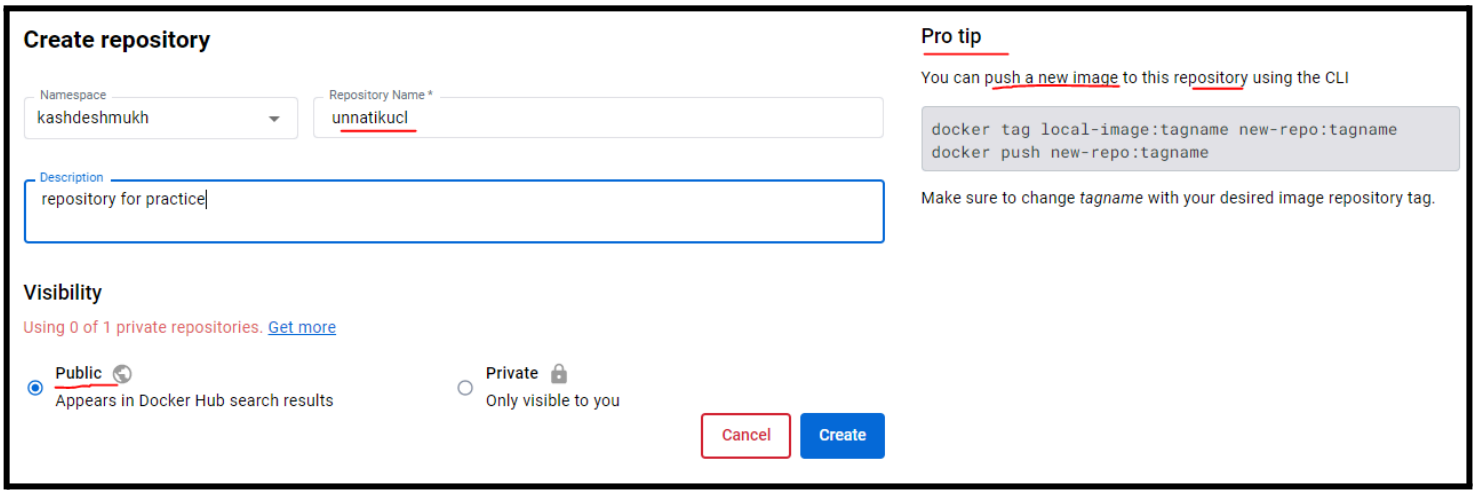
Before we can push an image to a container registry, we need to be logged in to that registry using the docker login command.
Create an account on **hub.docker.com => Registry (Public)**

$docker login

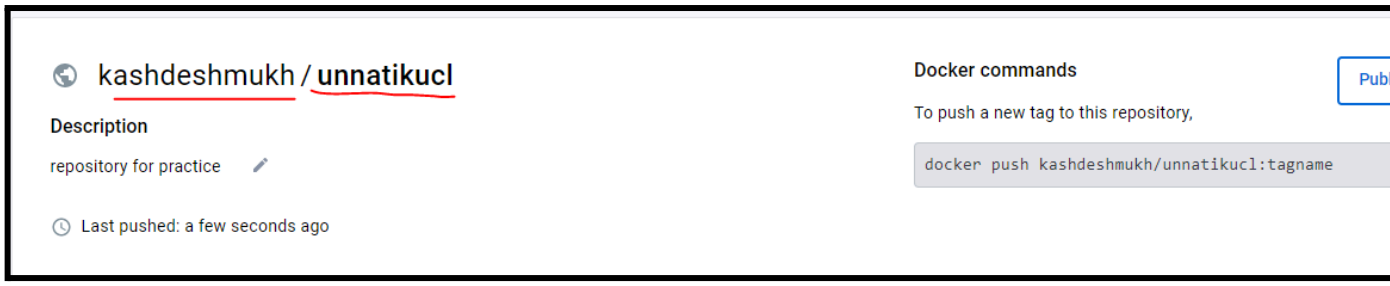
```
ubuntu $ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://h
ub.docker.com to create one.
Username: kashdeshmukh
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

On hub.docker.com; creating a **repository**.
Repositories can be **PUBLIC** or **PRIVATE**.
PUBLIC repository can be accessed by anyone.
PRIVATE repository is only accessed by specific people eg. employees in an organization.





We will be creating a public repository, with name “unnatikucl”
Here,
Kashdeshmukh = registry
Unnatikucl = repository



Therefore, a Registry is created. Repository is also created. Now it's time to create an Image.
For now, we will create a new image using an already created image.
Here, we will use **docker.io/httpd** image as our base image.

```
ubuntu $ docker pull docker.io/httpd
Using default tag: latest
latest: Pulling from library/httpd
f1f26f570256: Pull complete
a6b093ae1967: Pull complete
6b400bbb27df: Pull complete
d9833ead928a: Pull complete
ace056404ed3: Pull complete
Digest: sha256:f3e9eb9acace5bbc13c924293d2247a65bb59b8f062bcd98cd87ee4e18f86733
Status: Downloaded newer image for httpd:latest
docker.io/library/httpd:latest
```

To identify, where we need to push the image, we use “**tag**”

$docker tag <image> <registry_name>/<repo_name>:<image_name>

```
ubuntu $ docker tag httpd kashdeshmukh/unnatikucl:myimage1
ubuntu $
```

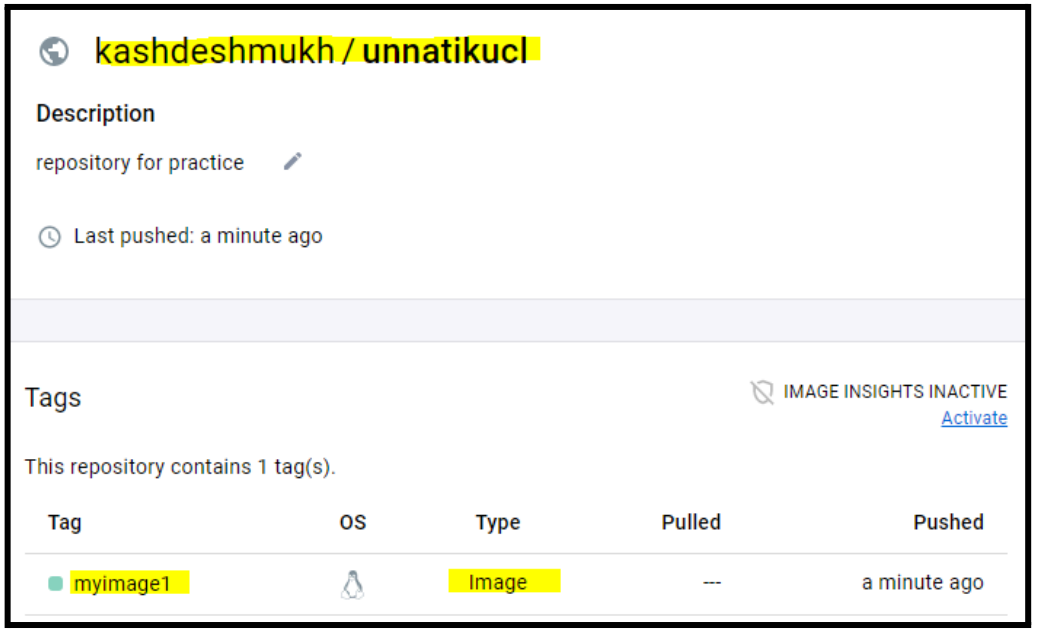
Here,
1 = kashdeshmukh = registry name
2 = unnatikucl = repository name
3 = myimage1 = image name
Thus image gets created:

```
ubuntu $ docker images
REPOSITORY                TAG         IMAGE ID       CREATED      SIZE
kashdeshmukh/unnatikucl   myimage1    192d41583429   5 days ago   145MB
httpd                     latest      192d41583429   5 days ago   145MB
ubuntu $
```

Now, the next step is to PUSH this image to the registry => repository:
$docker push <registry_name>/<repo_name>:<image_name>

```
ubuntu $ docker images
REPOSITORY                TAG         IMAGE ID       CREATED      SIZE
httpd                     latest      192d41583429   5 days ago   145MB
kashdeshmukh/unnatikucl   myimage1    192d41583429   5 days ago   145MB
ubuntu $ docker push kashdeshmukh/unnatikucl:myimage1
The push refers to repository [docker.io/kashdeshmukh/unnatikucl]
10f56d9b82f4: Mounted from library/httpd
626b836c6b3c: Mounted from library/httpd
8065d3aedc6d: Mounted from library/httpd
d0dbdb0bf1f7: Mounted from library/httpd
3af14c9a24c9: Mounted from library/httpd
myimage1: digest: sha256:57b550dabe7be9a0af9a5cad02636c050a43522241cb5c6048e87240b9c15001 size: 1366
```

Let's check in hub.docker.com



We can pull this image too, as it is public. Let's just remove the image from the local machine.

```
ubuntu $ docker images
REPOSITORY                TAG        IMAGE ID       CREATED      SIZE
kashdeshmukh/unnatikucl   myimage1   192d41583429   5 days ago   145MB
httpd                     latest     192d41583429   5 days ago   145MB
ubuntu $ docker rmi kashdeshmukh/unnatikucl
Error: No such image: kashdeshmukh/unnatikucl
ubuntu $ docker rmi kashdeshmukh/unnatikucl:myimage1
Untagged: kashdeshmukh/unnatikucl:myimage1
Untagged: kashdeshmukh/unnatikucl@sha256:57b550dabe7be9a0af9a5cad02636c050a43522241cb5c6048e87240b9c15001
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED      SIZE
httpd        latest    192d41583429   5 days ago   145MB
```

$docker pull <registry_name>/<repo_name>:<image_name>

```
ubuntu $ docker pull kashdeshmukh/unnatikucl:myimage1
myimage1: Pulling from kashdeshmukh/unnatikucl
Digest: sha256:57b550dabe7be9a0af9a5cad02636c050a43522241cb5c6048e87240b9c15001
Status: Downloaded newer image for kashdeshmukh/unnatikucl:myimage1
docker.io/kashdeshmukh/unnatikucl:myimage1
ubuntu $ docker images
REPOSITORY                TAG        IMAGE ID       CREATED      SIZE
kashdeshmukh/unnatikucl   myimage1   192d41583429   5 days ago   145MB
httpd                     latest     192d41583429   5 days ago   145MB
```

MICROSERVICE APPLICATIONS VS MONOLITHIC APPLICATIONS

Microservice and monolithic **architectures** are two different approaches to building applications. Here are some of the key differences between them:

Monolithic applications:

- In a monolithic architecture, the entire application is built as a single, cohesive unit.
- All of the components of the application are tightly integrated and run in the same process or on the same machine.
- Monolithic applications are typically easier to develop and deploy than microservice applications, since there is only one codebase to manage.
- However, monolithic applications can become difficult to maintain and scale as they grow larger and more complex.
- Adding new features or making changes to the application can require a full rebuild and redeployment of the entire application.

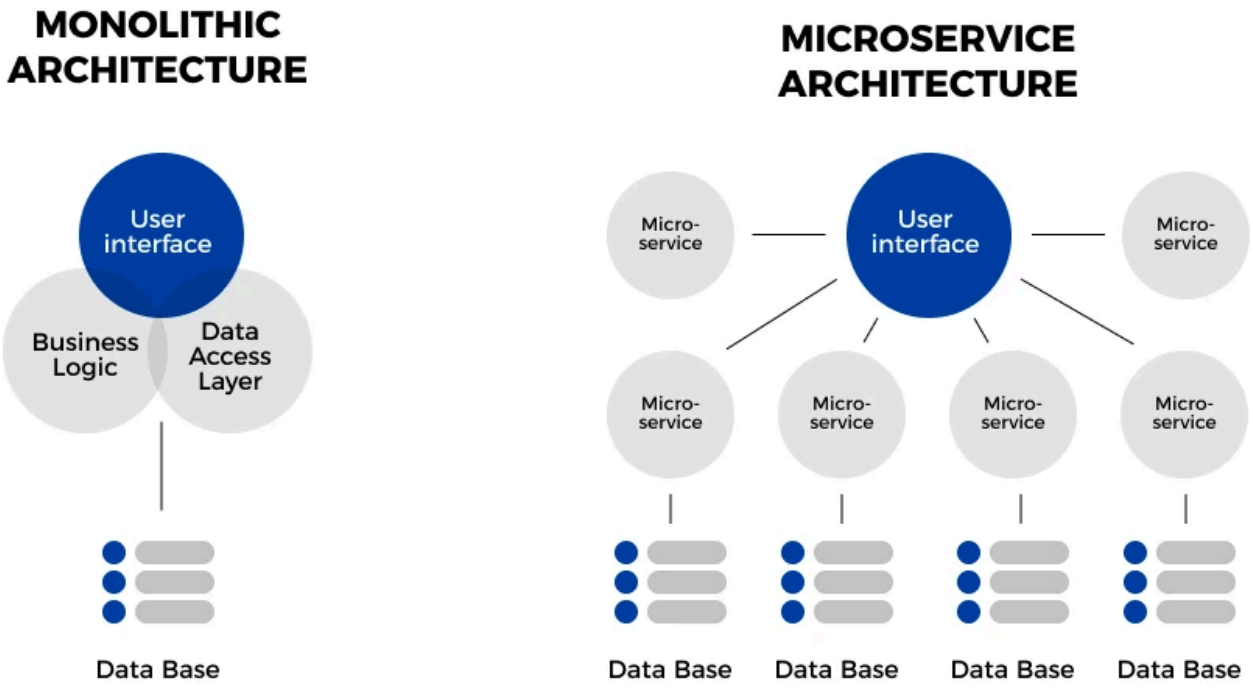
Microservice applications:

- In a microservice architecture, the application is broken down into smaller, independent services that communicate with each other over a network.
- Each service is responsible for a specific task or set of tasks.
- Microservices can be developed and deployed independently, which can make it easier to scale and update the application over time.

Thus, Microservice Application Architecture requires less resources, ultimately becoming light-weight. Thus, can be easily deployed on a docker container.

For deploying microservices application on container:

1. Write an image using DockerFile.
2. Add the application in the DockerFile.
3. Image is now created.
4. Deploy the container using this image.



DOCKER COMMIT / PRE-DOCKER FILE

The docker commit command in Docker is used to create a new Docker image based on changes made to an existing container.
When we use docker commit, Docker takes a snapshot of the current state of the container's file system and creates a new image from that snapshot.

The basic syntax for the docker commit command is as follows:
$docker commit <CONTAINER> <NEW_IMAGE_NAME>

The docker commit command can be useful for creating custom images based on existing images.
For example, we might start with a **base image** like **ubuntu**, make some changes to it by running commands in a container, and then use docker **commit** to create a new image based on those changes.
This new image will be stored on our **local machine** and can be used to create new containers in the future.

Practical:

Pulling the Base image : docker.io/ubuntu

```
ubuntu $ #DOCKER COMMIT
ubuntu $ #1.BASE IMAGE : docker.io/ubuntu
ubuntu $ docker pull docker.io/ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
2ab09b027e7f: Pull complete
Digest: sha256:67211c14fa74f070d27cc59d69a7fa9aeff8e28ea118ef3babc295a0428a6d21
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

Creating a docker container “test1” using our base image.
Making some changes in the container.

```
ubuntu $ docker run --name=test1 -it docker.io/ubuntu
root@62d9311779da:/# echo "learning Ubuntu Commit" > myfile1.txt
root@62d9311779da:/# exit
exit
```

Docker commit : this will change the Container into an Image.
$docker commit <container_name> <image_name>

```
ubuntu $ docker commit test1 myimage1
sha256:777024d03f43b7b6ed8b3fdb5d22b501bcd12b7860caa232940d94d3a3d81c83
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
myimage1     latest    777024d03f43   5 seconds ago    77.8MB
ubuntu       latest    08d22c0ceb15   3 weeks ago      77.8MB
```

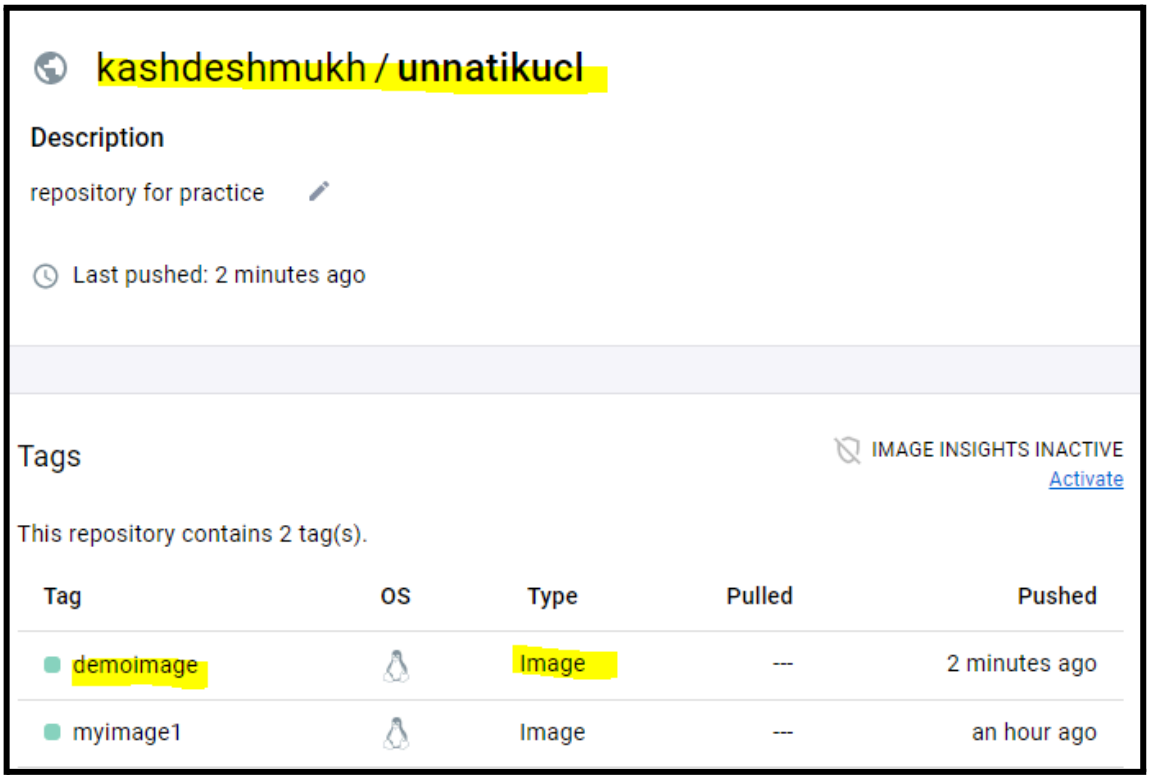
Running a new container using the new image: “myimage1”

```
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED              SIZE
myimage1     latest    777024d03f43   About a minute ago   77.8MB
ubuntu       latest    08d22c0ceb15   3 weeks ago          77.8MB
ubuntu $ docker run -it myimage1 bash
root@de88cdafc1a2:/# l
bin@   dev/  home/  lib32@  libx32@  mnt/         opt/   root/  sbin@  sys/  usr/
boot/  etc/  lib@   lib64@  media/   myfile1.txt  proc/  run/   srv/   tmp/  var/
root@de88cdafc1a2:/# cat myfile1.txt
learning Ubuntu Commit
root@de88cdafc1a2:/#
```

We can push this image to the registry.

```
ubuntu $ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://h
ub.docker.com to create one.
Username: kashdeshmukh
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
ubuntu $ docker tag myimage1 kashdeshmukh/unnatikucl:demoimage
ubuntu $ docker images
REPOSITORY                TAG         IMAGE ID       CREATED         SIZE
kashdeshmukh/unnatikucl   demoimage   777024d03f43   3 minutes ago   77.8MB
myimage1                  latest      777024d03f43   3 minutes ago   77.8MB
ubuntu                    latest      08d22c0ceb15   3 weeks ago     77.8MB
ubuntu $ docker push kashdeshmukh/unnatikucl:demoimage
The push refers to repository [docker.io/kashdeshmukh/unnatikucl]
545297043dc7: Pushed
b93c1bd012ab: Mounted from library/ubuntu
demoimage: digest: sha256:e2f5bdca86ce0f70e1751a4f4856fc4f7cc12d527d993938ba17531de3410863 size: 736
```

Let’s verify



But if we don't have any registry to upload / push this image; and we want to give the image to someone else; then:

We can use the "save" option to convert the image into tar file.
$docker save -o <filename.tar> <imagename>

```
ubuntu $ docker images
REPOSITORY                TAG         IMAGE ID       CREATED          SIZE
kashdeshmukh/unnatikucl   demoimage   777024d03f43   10 minutes ago   77.8MB
myimage1                  latest      777024d03f43   10 minutes ago   77.8MB
ubuntu                    latest      08d22c0ceb15   3 weeks ago      77.8MB
ubuntu $ docker save -o demo.tar myimage1
ubuntu $ ls
demo.tar  dsc_compose  dsc_streamlit  fhtw-hello-world  filesystem
```

Docker Save Options:

```
ubuntu $ docker --help save

Usage:  docker save [OPTIONS] IMAGE [IMAGE...]

Save one or more images to a tar archive (streamed to STDOUT by default)

Options:
  -o, --output string   Write to a file, instead of STDOUT
```

Now, we can share this tar file "demo.tar" to the person.

The person on his/her end will perform the following steps.
$docker load -i <filename>
This will load the tar file into the image.

```
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
ubuntu $ ls
demo.tar  dsc_compose  dsc_streamlit  fhtw-hello-world  filesystem
ubuntu $ docker load -i demo.tar
Loaded image: myimage1:latest
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
myimage1     latest    777024d03f43   14 minutes ago   77.8MB
```

Docker load Options:

```
ubuntu $ docker --help load

Usage:  docker load [OPTIONS]

Load an image from a tar archive or STDIN

Options:
  -i, --input string   Read from tar archive file, instead of STDIN
  -q, --quiet          Suppress the load output
```

DOCKER FILE

A **Dockerfile** is a text file that contains instructions for building a Docker image.
The instructions are Human-readable.
It has no programming / coding.

Let's create a Dockerfile.
Example 1:
Creating a directory "Unnati" and cd.
Creating a docker file inside "unnati" dir.
The docker file name, as per standard, is "**Dockerfile**". It is case-sensitive.

```
ubuntu $ mkdir unnati
ubuntu $ cd unnati
ubuntu $ pwd
/root/unnati
ubuntu $ vi Dockerfile
```

Inside DockerFile, we write the instructions:

FROM docker.io/ubuntu
=> This says that **docker.io/ubuntu** is our **base image.**
We will use this image and make some changes to it.

RUN echo "Learning DockerFile" > myfile.txt
=> This creates a file "myfile.txt" and writes "Learning DockerFile" into the file.

RUN mkdir kucl
=>This creates a directory named "kucl" in the container.

```
#Instruction 1: FROM
FROM docker.io/ubuntu

#Instruction 2: RUN
RUN echo "Learning Dockerfile" > myfile1
RUN mkdir kucl
~
```

After writing the Dockerfile, we need to **Build** the image.
$docker build -t <image_name> .

-t : tag
<image_name> : this name will be given to the image
. : current location.

It automatically takes the DockerFile as input for building the image, as it is the default / standard file.
For executing the instructions, a container is created at the backend, which is removed after the work is finished.

```
ubuntu $ docker build -t myimage .
Sending build context to Docker daemon  2.048kB
Step 1/3 : FROM docker.io/ubuntu
latest: Pulling from library/ubuntu
2ab09b027e7f: Pull complete
Digest: sha256:67211c14fa74f070d27cc59d69a7fa9aeff8e28ea118ef3babc295a0428a6d21
Status: Downloaded newer image for ubuntu:latest
 ---> 08d22c0ceb15
Step 2/3 : RUN echo "Learning Dockerfile" > myfile1
 ---> Running in ffa78c1cd704
Removing intermediate container ffa78c1cd704
 ---> 63eb20b64dc3
Step 3/3 : RUN mkdir kucl
 ---> Running in a924ab888018
Removing intermediate container a924ab888018
 ---> 6c58802aeea3
Successfully built 6c58802aeea3
Successfully tagged myimage:latest
```

Running a container using the created image

```
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
myimage      latest    6c58802aeea3   8 minutes ago   77.8MB
ubuntu       latest    08d22c0ceb15   3 weeks ago     77.8MB
ubuntu $ docker run --name=demo -it docker.io/myimage bash
root@dfa5a36b26ba:/# ls
bin   dev  home  lib    lib64   media  myfile1  proc  run   srv  tmp  var
boot  etc  kucl  lib32  libx32  mnt    opt      root  sbin  sys  usr
root@dfa5a36b26ba:/# cat myfile1
Learning Dockerfile
```

Example 2:

Inside directory “unnati”, creating a file “localfile” with some text inside it.

```
ubuntu $ mkdir unnati
ubuntu $ cd unnati
ubuntu $ ls
ubuntu $ pwd
/root/unnati
ubuntu $ echo "Hello World" > localfile
ubuntu $ ls
localfile
ubuntu $ cat localfile
Hello World
ubuntu $ vi Dockerfile
```

Writing instruction inside the Dockerfile.

COPY localfile /root/containerfile

=> COPY command will copy the file “localfile” which we created on the local machine, to the docker image.

However, COPY is only capable of copying files and directories from Local machine to the image in /root with the name “containerfile”.

The COPY command supports the “Absolute” path.

```
#Instruction 1: FROM
FROM docker.io/ubuntu

#Instruction 2: RUN
RUN echo "Learning DockerFIle" > file1

#Instruction 3: COPY
COPY localfile /root/containerfile
#This will copy the file "localfile" from local machine to the Image
```

Building the image with the name “**image1**”.

```
ubuntu $ docker build -t image1 .
Sending build context to Docker daemon  3.072kB
Step 1/3 : FROM docker.io/ubuntu
latest: Pulling from library/ubuntu
2ab09b027e7f: Pull complete
Digest: sha256:67211c14fa74f070d27cc59d69a7fa9aeff8e28ea118ef3babc295a0428a6d21
Status: Downloaded newer image for ubuntu:latest
 ---> 08d22c0ceb15
Step 2/3 : RUN echo "Learning DockerFIle" > file1
 ---> Running in 36481f3f5ef3
Removing intermediate container 36481f3f5ef3
 ---> 7f7993958f4a
Step 3/3 : COPY localfile /root/containerfile
 ---> 91c97cd03af2
Successfully built 91c97cd03af2
Successfully tagged image1:latest
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
image1       latest    91c97cd03af2   5 seconds ago    77.8MB
ubuntu       latest    08d22c0ceb15   3 weeks ago      77.8MB
```

Let's run a container using "image1"

```
ubuntu $ docker run --name container1 -it image1 bash
root@e224b8a0e062:/# ls
bin   dev   file1  lib    lib64   media  opt   root  sbin  sys  usr
boot  etc   home   lib32  libx32  mnt    proc  run   srv   tmp  var
root@e224b8a0e062:/# cat file1
Learning DockerFIle
root@e224b8a0e062:/# ls /
bin   dev   file1  lib    lib64   media  opt   root  sbin  sys  usr
boot  etc   home   lib32  libx32  mnt    proc  run   srv   tmp  var
root@e224b8a0e062:/# ls /root/
containerfile
root@e224b8a0e062:/# cat /root/containerfile
Hello World
root@e224b8a0e062:/# exit
exit
ubuntu $
```

Example 3:

```
ubuntu $ ls
filesystem  task  unnati
ubuntu $ cd unnati/
ubuntu $ ls
Dockerfile  localfile
ubuntu $ vi Dockerfile
```

Writing the Dockerfile

ADD https://1000logos.net/wp-content/uploads/2021/11/Docker-Logo-2013.png /root/dockerlogo.png
=> This command will take the logo from google, i.e. a remote system, and then copy it to the image.
Thus ADD instruction is used to copy files from remote system as well as local system to the docker image.

```
#Instruction 1: FROM
FROM docker.io/ubuntu

#Instruction 2: RUN
RUN echo "Learning DockerFIle" > file1

#Instruction 3: COPY
COPY localfile /root/containerfile
#This will copy the file "localfile" from local machine to the Image

#Instruction 4: ADD
ADD https://1000logos.net/wp-content/uploads/2021/11/Docker-Logo-2013.png /root/dockerlogo.png
#This will copy the docker from remote to the Image
~
```

Building the docker image

```
ubuntu $ docker build -t myimage .
Sending build context to Docker daemon  3.072kB
Step 1/4 : FROM docker.io/ubuntu
 ---> 08d22c0ceb15
Step 2/4 : RUN echo "Learning DockerFIle" > file1
 ---> Running in 39fbb5bee520
Removing intermediate container 39fbb5bee520
 ---> 31c6139150c6
Step 3/4 : COPY localfile /root/containerfile
 ---> f372ee43f7e1
Step 4/4 : ADD https://1000logos.net/wp-content/uploads/2021/11/Docker-Logo-2013.png /root/dockerlogo.png
Downloading [==================================================>]  57.19kB/57.19kB

 ---> 92ac392a5473
Successfully built 92ac392a5473
Successfully tagged myimage:latest
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
myimage      latest    92ac392a5473   6 seconds ago    77.9MB
taskimage    latest    c096fdfb6c6f   10 minutes ago   77.8MB
ubuntu       latest    08d22c0ceb15   3 weeks ago      77.8MB
```

Running a container “test” using the image.

```
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
myimage      latest    92ac392a5473   57 seconds ago   77.9MB
taskimage    latest    c096fdfb6c6f   11 minutes ago   77.8MB
ubuntu       latest    08d22c0ceb15   3 weeks ago      77.8MB
ubuntu $ docker run --name test -it myimage bash
root@d8615fe968dd:/# cd /root/
root@d8615fe968dd:~# ls
containerfile  dockerlogo.png
root@d8615fe968dd:~#
```

TASK

Create a Directory on the local machine, create some files inside it and COPY those to the image.

```
ubuntu $ mkdir task
ubuntu $ ls
filesystem  task  unnati
ubuntu $ cd task
ubuntu $ pwd
/root/task
ubuntu $ mkdir dir1
ubuntu $ cd dir1
ubuntu $ echo "Hi, This is file1" > file1
ubuntu $ echo "Hi, This is file2" > file2
ubuntu $ echo "Hi, This is file3" > file3
ubuntu $ ls
file1  file2  file3
ubuntu $ cd ..
ubuntu $ pwd
/root/task
ubuntu $ vi Dockerfile
```

Writing Instructions in Dockerfile

```
FROM docker.io/ubuntu

COPY dir1/ /root/mydir
```

Building the image:

```
ubuntu $ docker build -t taskimage .
Sending build context to Docker daemon  5.632kB
Step 1/2 : FROM docker.io/ubuntu
 ---> 08d22c0ceb15
Step 2/2 : COPY dir1 /root/
 ---> 1d5794495338
Successfully built 1d5794495338
Successfully tagged taskimage:latest
```

Running the container “demo” using the image

```
ubuntu $ docker build -t taskimage .
Sending build context to Docker daemon  5.632kB
Step 1/2 : FROM docker.io/ubuntu
 ---> 08d22c0ceb15
Step 2/2 : COPY dir1/ /root/mydir
 ---> c096fdfb6c6f
Successfully built c096fdfb6c6f
Successfully tagged taskimage:latest
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
taskimage    latest    c096fdfb6c6f   6 seconds ago   77.8MB
ubuntu       latest    08d22c0ceb15   3 weeks ago     77.8MB
ubuntu $ docker run --name demo -it taskimage bash
root@70fec38f498f:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@70fec38f498f:/# cd /root/
root@70fec38f498f:~# ls
mydir
root@70fec38f498f:~# cd mydir/
root@70fec38f498f:~/mydir# ls
file1  file2  file3
root@70fec38f498f:~/mydir# exit
exit
```

```
ubuntu $ docker run --name demo1 -it taskimage bash
root@247069335ed7:/# cd /root/mydir/
root@247069335ed7:~/mydir# ls
file1  file2  file3
root@247069335ed7:~/mydir# cat file1
Hi, This is file1
root@247069335ed7:~/mydir# cat file2
Hi, This is file2
root@247069335ed7:~/mydir# cat file3
Hi, This is file3
root@247069335ed7:~/mydir#
```

Example 4:

Creating a directory “unnati”.
Creating a docker file **“docf1”** inside the directory.
This time, we are not giving the standard Dockerfile name.

```
ubuntu $ mkdir unnati
ubuntu $ cd unnati
ubuntu $ pwd
/root/unnati
ubuntu $ vi docf1
```

Writing instructions in the dockerfile “docf1”

```
FROM docker.io/busybox

RUN echo "testing" > /root/myfile1
```

Building the docker image.
As we have not used the standard docker file, therefore we will be using the option “**-f**” followed by file name and . for the current location.

```
ubuntu $ docker build -t test1 -f docf1 .
Sending build context to Docker daemon  2.048kB
Step 1/2 : FROM docker.io/busybox
latest: Pulling from library/busybox
4b35f584bb4f: Pull complete
Digest: sha256:b5d6fe0712636ceb7430189de28819e195e8966372edfc2d9409d79402a0dc16
Status: Downloaded newer image for busybox:latest
 ---> 7cfbbec8963d
Step 2/2 : RUN echo "testing" > /root/myfile1
 ---> Running in c6dad728ac56
Removing intermediate container c6dad728ac56
 ---> 19613e8310b3
Successfully built 19613e8310b3
Successfully tagged test1:latest
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
test1        latest    19613e8310b3   3 seconds ago    4.86MB
busybox      latest    7cfbbec8963d   12 days ago      4.86MB
```

Running a Container using the image.
We will write a startup script that will be executed after the container is created.
Thus, whatever we write after the <image_name> in docker run command, will be treated as startup script and will be executed.

```
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
test1        latest    19613e8310b3   2 minutes ago    4.86MB
busybox      latest    7cfbbec8963d   12 days ago      4.86MB
ubuntu $ #Writing startup script in the docker run command
ubuntu $ docker run test1 /bin/ls
bin
dev
etc
home
lib
lib64
proc
root
sys
tmp
usr
var
```

Pinging the google.com

```
ubuntu $ docker run test1 ping google.com
PING google.com (172.217.194.102): 56 data bytes
64 bytes from 172.217.194.102: seq=0 ttl=51 time=1.475 ms
64 bytes from 172.217.194.102: seq=1 ttl=51 time=1.605 ms
64 bytes from 172.217.194.102: seq=2 ttl=51 time=1.439 ms
64 bytes from 172.217.194.102: seq=3 ttl=51 time=1.509 ms
```

When we write -d in docker run, it will be executed in the background

```
ubuntu $ docker run -d test1 uname -a
49fcc97e9944a0979f8157cc2c359af7b3e709fb036334879c893ecb6fd1159e
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ubuntu $ docker ps -a
CONTAINER ID   IMAGE     COMMAND            CREATED              STATUS                          PORTS     NAMES
49fcc97e9944   test1     "uname -a"         16 seconds ago       Exited (0) 15 seconds ago                 vibrant_shir
ley
b19e8a78d0d4   test1     "ping google.com"  About a minute ago   Exited (0) About a minute ago             charming_hof
stadter
2f61ccd4d8e7   test1     "/bin/ls"          4 minutes ago        Exited (0) 4 minutes ago                  infallible_w
illiamson
ubuntu $ docker logs 49f
Linux 49fcc97e9944 5.4.0-131-generic #147-Ubuntu SMP Fri Oct 14 17:07:22 UTC 2022 x86_64 GNU/Linux
ubuntu $ docker logs b19
PING google.com (172.217.194.102): 56 data bytes
64 bytes from 172.217.194.102: seq=0 ttl=51 time=1.475 ms
64 bytes from 172.217.194.102: seq=1 ttl=51 time=1.605 ms
64 bytes from 172.217.194.102: seq=2 ttl=51 time=1.439 ms
64 bytes from 172.217.194.102: seq=3 ttl=51 time=1.509 ms

--- google.com ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 1.439/1.507/1.605 ms
```

Pinging google.com , on a container in the background

```
ubuntu $ docker run -d test1 ping google.com
f56d7a775aa5d0a6e08c3903d34adc3a6ea36b7b17b5a71922570e32f1be5b83
ubuntu $ docker logs f56
PING google.com (172.217.194.139): 56 data bytes
64 bytes from 172.217.194.139: seq=0 ttl=100 time=1.715 ms
64 bytes from 172.217.194.139: seq=1 ttl=100 time=1.663 ms
64 bytes from 172.217.194.139: seq=2 ttl=100 time=1.694 ms
64 bytes from 172.217.194.139: seq=3 ttl=100 time=1.743 ms
64 bytes from 172.217.194.139: seq=4 ttl=100 time=1.743 ms
64 bytes from 172.217.194.139: seq=5 ttl=100 time=1.765 ms
64 bytes from 172.217.194.139: seq=6 ttl=100 time=1.703 ms
64 bytes from 172.217.194.139: seq=7 ttl=100 time=1.678 ms
64 bytes from 172.217.194.139: seq=8 ttl=100 time=1.730 ms
64 bytes from 172.217.194.139: seq=9 ttl=100 time=1.771 ms
64 bytes from 172.217.194.139: seq=10 ttl=100 time=1.791 ms
64 bytes from 172.217.194.139: seq=11 ttl=100 time=1.740 ms
64 bytes from 172.217.194.139: seq=12 ttl=100 time=1.717 ms
```

Example 5:

Creating a directory “unnati”, and creating a docker file “docf1” inside it.

```
ubuntu $ mkdir unnati
ubuntu $ cd unnati
ubuntu $ ls
ubuntu $ vi docf1
```

Writing instructions inside the docker file.

```
FROM docker.io/ubuntu

#commands must be non-interactive . Therefore "-y"
RUN apt update -y

#install the ping command inside the image
RUN apt install iputils-ping -y
```

Building the docker image

```
ubuntu $ docker build -t myimage -f docf1 .
Sending build context to Docker daemon  2.048kB
Step 1/3 : FROM docker.io/ubuntu
latest: Pulling from library/ubuntu
2ab09b027e7f: Pull complete
Digest: sha256:67211c14fa74f070d27cc59d69a7fa9aeff8e28ea118ef3babc295a0428a6d21
Status: Downloaded newer image for ubuntu:latest
 ---> 08d22c0ceb15
Step 2/3 : RUN apt update -y
 ---> Running in acc21e2a668a
```

Running the docker container using the image

```
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED              SIZE
myimage      latest    880e4283db89   About a minute ago   122MB
ubuntu       latest    08d22c0ceb15   3 weeks ago          77.8MB
ubuntu $ docker run myimage ping google.com
PING google.com (172.217.194.100) 56(84) bytes of data.
64 bytes from si-in-f100.1e100.net (172.217.194.100): icmp_seq=1 ttl=100 time=1.33 ms
64 bytes from si-in-f100.1e100.net (172.217.194.100): icmp_seq=2 ttl=100 time=1.39 ms
64 bytes from si-in-f100.1e100.net (172.217.194.100): icmp_seq=3 ttl=100 time=1.37 ms
64 bytes from si-in-f100.1e100.net (172.217.194.100): icmp_seq=4 ttl=100 time=1.42 ms
```

Example 6:

Creating a directory “unnati”

Creating a file “Dockerfile” inside it.

Writing the instructions in the Dockerfile.

```
FROM docker.io/centos:7

#Instruction 5: MAINTAINER
#Basically this will assign unnatikucl@gmail.com as the owner of the dockerfile
MAINTAINER unnatikucl@gmail.com

#install httpd
RUN yum install httpd -y

#write content in defalt httpd file
RUN echo "hello from httpd" > /var/www/html/index.html
```

Building the docker image

```
ubuntu $ docker build -t myimage -f docf1 .
Sending build context to Docker daemon  2.048kB
Step 1/3 : FROM docker.io/ubuntu
latest: Pulling from library/ubuntu
2ab09b027e7f: Pull complete
Digest: sha256:67211c14fa74f070d27cc59d69a7fa9aeff8e28ea118ef3babc295a0428a6d21
Status: Downloaded newer image for ubuntu:latest
 ---> 08d22c0ceb15
Step 2/3 : RUN apt update -y
 ---> Running in acc21e2a668a
```

Creating the container using the image.

Here, **"/usr/sbin/httpd -D FOREGROUND" is used to start httpd in centos.**

```
ubuntu $ docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
myimage1      latest    f00daa725c55   4 minutes ago    440MB
centos        7         eeb6ee3f44bd   18 months ago    204MB
ubuntu $ docker run -d --name=cont1 myimage1 /usr/sbin/httpd -D FOREGROUND
d702d125d9e54ea7e1c22d6db9a285db679e30ae158f55be00885b63840fb33c
ubuntu $ docker exec -it cont1 bash
[root@d702d125d9e5 /]# cd /var/www/html/
[root@d702d125d9e5 html]# ls
index.html
[root@d702d125d9e5 html]# cat index.html
hello from httpd
[root@d702d125d9e5 html]#
```

Example 7:

Creating 2 files on a local machine.

a. Index.html

b. myfile1

```
ubuntu $ pwd
/root/unnati
ubuntu $ echo "Hello World" > myfile1
ubuntu $ echo "This is my index.html file on local machine" > index.html
ubuntu $ ls
Dockerfile  docf1  index.html  myfile1
ubuntu $ vi Dockerfile
```

Writing the docker instructions in dockerfile

```
FROM docker.io/centos:7

#Instruction 5: MAINTAINER
#Basically this will assign unnatikucl@gmail.com as the owner of the dockerfile
MAINTAINER unnatikucl@gmail.com

#Instruction 6: WORKDIR
#changes the present working directory
WORKDIR /var/www/html

#install httpd
RUN yum install httpd -y

#Copying the file index.html and myfile1 from local machine to the image in pwd
COPY index.html .
COPY myfile1 .
```

Building the docker image

```
ubuntu $ docker build -t myhttpdimage .
Sending build context to Docker daemon   5.12kB
Step 1/6 : FROM docker.io/centos:7
 ---> eeb6ee3f44bd
Step 2/6 : MAINTAINER unnatikucl@gmail.com
 ---> Using cache
 ---> 89f1094ddd32
Step 3/6 : WORKDIR /var/www/html
 ---> Running in df4159375d05
Removing intermediate container df4159375d05
 ---> f737842b6a04
Step 4/6 : RUN yum install httpd -y
```

Running a container “cont” on the image

```
ubuntu $ docker images
REPOSITORY     TAG       IMAGE ID       CREATED          SIZE
myhttpdimage   latest    c31136d972c4   3 minutes ago    440MB
<none>         <none>    f00daa725c55   17 minutes ago   440MB
centos         7         eeb6ee3f44bd   18 months ago    204MB
ubuntu $ docker run -d --name=cont myhttpdimage /usr/sbin/httpd -D FOREGROUND
40bb377b531d400de1ebef3934929e3059707e3df09bb2ae281e0b50374e216c
ubuntu $ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS     NAMES
40bb377b531d   myhttpdimage   "/usr/sbin/httpd -D …"   3 seconds ago   Up 2 seconds             cont
ubuntu $ docker exec -it cont bash
[root@40bb377b531d html]# pwd
/var/www/html
[root@40bb377b531d html]# ls
index.html  myfile1
[root@40bb377b531d html]# cat index.html
This is my index.html file on local machine
[root@40bb377b531d html]# cat myfile1
Hello World
[root@40bb377b531d html]#
```

Note:
RUN instruction is used while Building the image.
CMD instruction is used when a docker container is run.

Example 8:

Creating a directory “unnati”.
Creating a file index.html and writing some content in it.
Creating a file “Dockerfile” inside “unnati”.
Writing the instructions in the docker file.

```
FROM docker.io/centos:7

RUN yum install httpd -y

COPY index.html /var/www/html

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

Building the Docker image:

```
ubuntu $ docker build -t myimage .
Sending build context to Docker daemon  3.072kB
Step 1/4 : FROM docker.io/centos:7
 ---> eeb6ee3f44bd
Step 2/4 : RUN yum install httpd -y
 ---> Using cache
 ---> 6bf4e35b6110
Step 3/4 : COPY index.html /var/www/html
 ---> Using cache
 ---> 09ff44b0409f
Step 4/4 : CMD ["/usr/sbin/httpd","-D","FOREGROUND"]
 ---> Using cache
 ---> 2f55fef7ccfb
Successfully built 2f55fef7ccfb
Successfully tagged myimage:latest
```

Running a docker container using the image.

```
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
myimage      latest    2f55fef7ccfb   37 seconds ago   440MB
centos       7         eeb6ee3f44bd   18 months ago    204MB
ubuntu $ docker run -d --name=myapp myimage
791b14f9c1e047c2e84261cc1b03b79f62621c29f256d9ce3b4707a4646cf90b
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS     NAMES
791b14f9c1e0   myimage   "/usr/sbin/httpd -D …"   8 seconds ago   Up 6 seconds             myapp
```

$docker inspect myapp

```
"Env": [
    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
],
"Cmd": [
    "/usr/sbin/httpd",
    "-D",
    "FOREGROUND"
```

Therefore, no need to write the startup script manually. CMD will automatically run the startup script.

NOTE:
CMD used alone **can't be parameterized.**
But, **CMD** used along with **ENTRYPOINT** can be parameterized.
ENTRYPOINT remains constant.
It takes CMD as its parameter.

Example 9

Writing instructions in Dockerfile:

```
FROM docker.io/busybox

#Entrypoint instruction : this will remain constant.
ENTRYPOINT ["ping"]

#CMD instruction: CMD is the parameter to entrypoint. Here default parameter is "google.com",
#but can be changed while running the container.
CMD ["google.com"]
```

Building the docker image:

```
ubuntu $ docker build -t myimage .
Sending build context to Docker daemon  3.072kB
Step 1/3 : FROM docker.io/busybox
latest: Pulling from library/busybox
4b35f584bb4f: Pull complete
Digest: sha256:b5d6fe0712636ceb7430189de28819e195e8966372edfc2d9409d79402a0dc16
Status: Downloaded newer image for busybox:latest
 ---> 7cfbbec8963d
Step 2/3 : ENTRYPOINT ["ping"]
 ---> Running in cb68b8d9fb33
Removing intermediate container cb68b8d9fb33
 ---> 3580efdb82ae
Step 3/3 : CMD ["google.com"]
 ---> Running in 4fa1e0676ca5
Removing intermediate container 4fa1e0676ca5
 ---> 1d99d4565416
Successfully built 1d99d4565416
Successfully tagged myimage:latest
```

Running a docker container using the image.
#Not specifying any parameter. Therefore, pinging the default parameter: "google.com"

```
ubuntu $ docker run myimage
PING google.com (172.217.194.102): 56 data bytes
64 bytes from 172.217.194.102: seq=0 ttl=51 time=1.718 ms
64 bytes from 172.217.194.102: seq=1 ttl=51 time=1.676 ms
64 bytes from 172.217.194.102: seq=2 ttl=51 time=1.640 ms
64 bytes from 172.217.194.102: seq=3 ttl=51 time=1.694 ms
64 bytes from 172.217.194.102: seq=4 ttl=51 time=1.702 ms
^C
--- google.com ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 1.640/1.686/1.718 ms
```

Giving “yahoo.com” as a parameter, it will override google.com and ping yahoo.com

```
ubuntu $ docker run myimage yahoo.com
PING yahoo.com (74.6.231.20): 56 data bytes
64 bytes from 74.6.231.20: seq=0 ttl=43 time=210.284 ms
64 bytes from 74.6.231.20: seq=1 ttl=43 time=211.913 ms
64 bytes from 74.6.231.20: seq=2 ttl=43 time=210.383 ms
64 bytes from 74.6.231.20: seq=3 ttl=43 time=210.389 ms
^C
--- yahoo.com ping statistics ---
5 packets transmitted, 4 packets received, 20% packet loss
round-trip min/avg/max = 210.284/210.742/211.913 ms
```

RUNNING A SHELL SCRIPT USING DOCKERFILE

Example 10:

Creating a script file: “welcome.sh”

```
ubuntu $ vi welcome.sh
ubuntu $ cat welcome.sh
#welcome.sh script file

#!/bin/bash

echo "************************"

echo "Hello!! Welcome to KuCl"

echo "************************"
ubuntu $ sh welcome.sh
************************
Hello!! Welcome to KuCl
************************
```

Creating a dockerfile, and writing instructions in it

```
FROM docker.io/ubuntu

COPY welcome.sh /root/welcome.sh

#running the script when the container will be created
CMD ["sh","/root/welcome.sh"]
```

Building the docker image and creating a docker container using the image.
It will directly run the welcome.sh script.

```
ubuntu $ docker build -t myimage .
Sending build context to Docker daemon  4.096kB
Step 1/3 : FROM docker.io/ubuntu
latest: Pulling from library/ubuntu
2ab09b027e7f: Pull complete
Digest: sha256:67211c14fa74f070d27cc59d69a7fa9aeff8e28ea118ef3babc295a0428a6d21
Status: Downloaded newer image for ubuntu:latest
 ---> 08d22c0ceb15
Step 2/3 : COPY welcome.sh /root/welcome.sh
 ---> ebb8a7bebb13
Step 3/3 : CMD ["sh","/root/welcome.sh"]
 ---> Running in 606869ceb410
Removing intermediate container 606869ceb410
 ---> 3314a663541f
Successfully built 3314a663541f
Successfully tagged myimage:latest
ubuntu $ docker run myimage
**********************
Hello!! Welcome to KuCl
**********************
```

TASK:

Create 2 shell scripts: welcome.sh and thankyou.sh.
Create a parameterized container using ENTRYPOINT.

Creating 2 shell scripts.

```
ubuntu $ ls
Dockerfile  thankyou.sh  welcome.sh
ubuntu $ cat welcome.sh
#welcome.sh script file

#!/bin/bash

echo "**********************"

echo "Hello!! Welcome to KuCl"

echo "**********************"
ubuntu $ cat thankyou.sh
#thankyou.sh script

#!/bin/bash

echo "*******************************"
echo "Byee!!!Thankyou for visiting!!!"
echo "*******************************"
```

Writing Dockerfile

```
FROM docker.io/ubuntu

WORKDIR /root

COPY welcome.sh welcome.sh
COPY thankyou.sh thankyou.sh

#entrypoint: that will remain constant
ENTRYPOINT ["sh"]

#running the script when the container will be created
#CMD as a parameter to the container: by default: "welcome.sh"
CMD ["welcome.sh"]
```

Building the docker image and running the container:
a. Without parameter: **$docker run myimage**
b. With parameter: **$docker run myimage thankyou.sh**

```
ubuntu $ docker run myimage
**********************
Hello!! Welcome to KuCl
**********************
ubuntu $ docker run myimage thankyou.sh
*******************************
Byee!!!Thankyou for visiting!!!
*******************************
```

Example 11:

Creating a python file: welcome.py

```
#Python file: welcome.py

print("*"*14)
print("Hello World!!")
print("*"*14)
```

Creating a Dockerfile, and writing some instructions, which will automatically run the welcome.py script when the container is run.

```
FROM docker.io/ubuntu

RUN apt update -y
RUN apt install python3 -y

WORKDIR /root

COPY welcome.py welcome.py

CMD ["python3","welcome.py"]
```

Building the docker image, and running the container:

```
ubuntu $ docker run myimage
**************
Hello World!!
**************
```

NOTE:
echo $?
If the previous command is executed successfully, then echo $? Will give output '0'
But if not executed properly, then the output won't be '0'.

Example 12

Creating a Dockerfile and writing instructions in it.

```
FROM docker.io/ubuntu

#Environment variable
ENV mydir=docker

RUN mkdir /$mydir

RUN echo "hello" > /$mydir/myfile.txt

RUN echo "world" > /$mydir/mytest.txt
```

When we run the container, and inspect it, we can see the env variable with its value. Thus, env can be very useful and can prevent tedious and repetitive tasks

```
"Env": [
    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
    "mydir=docker"
```

Example 13:

Instructions in Dockerfile

```
FROM docker.io/ubuntu

#Environment variable
ENV mydir=docker

ENV port=8888

RUN apt update -y
RUN apt install apache2 -y

COPY index.html /var/www/html/index.html

#The EXPOSE instruction informs Docker that the container listens on the specified network ports at runtime.
EXPOSE $port

CMD ["/usr/sbin/apache2ctl","-D","FOREGROUND"]
```

Index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Welcome!!</title>
  </head>
  <body>
    <h1>Hello World</h1>
    <p>Hello From Apache</p>
  </body>
</html>
```

After inspecting the docker container:

```
"ExposedPorts": {
    "8888/tcp": {}
},
"Tty": false,
"OpenStdin": false,
"StdinOnce": false,
"Env": [
    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
    "mydir=docker",
    "port=8888"
]
```

```
root@a9036d0042a7:/# cat /var/www/html/index.html
<!DOCTYPE html>
<html>
  <head>
    <title>Welcome!!</title>
  </head>
  <body>
    <h1>Hello World</h1>
    <p>Hello From Apache</p>
  </body>
</html>
```

Example 14:

Writing instructions in Dockerfile

```
FROM docker.io/ubuntu

VOLUME /var/www/html

RUN useradd myuser1

USER myuser1
#Whenever we enter in the container, it will login using the myuser1.
```

Building the image, and creating a container.

When we enter the container, we will be logged in as “myuser1”.

```
ubuntu $ docker run -it myimage bash
myuser1@2d32f77a0a35:/$ id
uid=1000(myuser1) gid=1000(myuser1) groups=1000(myuser1)
```

NOTE: Expose and Volume : both are for **user information.**

Example 15:
Instruction: ARG

Dockerfile

```
FROM docker.io/ubuntu

#Instruction: ARG
ARG mydir somedir
#here, mydir is key and somedir is value. It is okay if we don't mention somedir.
#mydir will ultimately gets its value while building the image

WORKDIR $mydir
```

Building the docker image
For specifying the value of the arg key “mydir” we use:
$ docker build --build-arg <keyname>=<value> -t <image_name> .

```
ubuntu $ docker build --build-arg mydir=/tmp -t myimage .
Sending build context to Docker daemon  2.048kB
Step 1/3 : FROM docker.io/ubuntu
latest: Pulling from library/ubuntu
2ab09b027e7f: Pull complete
Digest: sha256:67211c14fa74f070d27cc59d69a7fa9aeff8e28ea118ef3babc295a0428a6d21
Status: Downloaded newer image for ubuntu:latest
 ---> 08d22c0ceb15
Step 2/3 : ARG mydir somedir
 ---> Running in c82f01f4834a
Removing intermediate container c82f01f4834a
 ---> db0ec6649a93
Step 3/3 : WORKDIR $mydir
 ---> Running in 4401e773f907
Removing intermediate container 4401e773f907
 ---> efcae0d11747
Successfully built efcae0d11747
Successfully tagged myimage:latest
ubuntu $ docker run -it myimage bash
root@9e8c46636146:/tmp#
```

Example 16:
Create 2 files “file1” and “file2” on the local system.
Create DockerFile and build the image.

```
ubuntu $ docker build --build-arg src=file1 -t myimage .
Sending build context to Docker daemon  4.096kB
Step 1/3 : FROM docker.io/ubuntu
 ---> 08d22c0ceb15
Step 2/3 : ARG src
 ---> Using cache
 ---> 8ed2dd196c91
Step 3/3 : COPY $src /tmp
 ---> d77c32fc1335
Successfully built d77c32fc1335
Successfully tagged myimage:latest
ubuntu $ docker run -it myimage bash
root@ff8dfc00b17c:/# ls /tmp/
file1
root@ff8dfc00b17c:/# exit
exit
ubuntu $ docker build --build-arg src=file2 -t myimage .
Sending build context to Docker daemon  4.096kB
Step 1/3 : FROM docker.io/ubuntu
 ---> 08d22c0ceb15
Step 2/3 : ARG src
 ---> Using cache
 ---> 8ed2dd196c91
Step 3/3 : COPY $src /tmp
 ---> 48d5337a9ae8
Successfully built 48d5337a9ae8
Successfully tagged myimage:latest
ubuntu $ docker run -it myimage bash
root@542fd6d812c3:/# ls /tmp/
file2
```

TASK

Write a Dockerfile, build its image.

It must have 1 or multiple arguments.

1 dockerfile to install httpd and must have welcome page

2 dockerfile to install nginx and must have a welcome page.

WILDFLY

Installing Wildfly manually

1. WildFly is an open source Java application server developed by Red Hat. Thus it is OpenSource.
2. It is a replacement for JBoss Application Server and is designed to be lightweight, fast, and flexible.
3. JBoss = downstream
   WildFly = upstream
4. WildFly is a popular choice for enterprise applications due to its ability to handle large workloads, scalability, and high availability features.
5. It also includes features such as clustering, load balancing, and distributed caching, which make it suitable for deployment in distributed environments.

Note:
WildFly, by default, runs on 8080 => Public (end-user)
Port number for management console => 9990

1. Downloading wildfly using wget.

```
root@rhel:~# wget https://github.com/wildfly/wildfly/releases/download/28.0.0.Beta1/wildfly-28.0.0.Beta1.zip
--2023-04-05 10:13:53--  https://github.com/wildfly/wildfly/releases/download/28.0.0.Beta1/wildfly-28.0.0.Beta1.zip
Resolving github.com (github.com)... 140.82.121.4
Connecting to github.com (github.com)|140.82.121.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65be/764708/df1fcf32-2d69-4854-97bb-4668f57186a5?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential
=AKIAIWNJYAX4CSVEH53A%2F20230405%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20230405T101353Z&X-Amz-Expires=300&X-Amz-Signature=1b129d022ee0bebef94ef50373e16243c61f7b44a778325afbb63
72347bd8e43&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=764708&response-content-disposition=attachment%3B%20filename%3Dwildfly-28.0.0.Beta1.zip&response-content-type=appli
cation%2Foctet-stream [following]
--2023-04-05 10:13:53--  https://objects.githubusercontent.com/github-production-release-asset-2e65be/764708/df1fcf32-2d69-4854-97bb-4668f57186a5?X-Amz-Algorithm=AWS4-HMAC-SHA256&X
-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20230405%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20230405T101353Z&X-Amz-Expires=300&X-Amz-Signature=1b129d022ee0bebef94ef50373e16243c61f7b
44a778325afbb6372347bd8e43&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=764708&response-content-disposition=attachment%3B%20filename%3Dwildfly-28.0.0.Beta1.zip&response-con
tent-type=application%2Foctet-stream
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.109.133, 185.199.108.133, 185.199.111.133, ...
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.109.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 235488103 (225M) [application/octet-stream]
Saving to: ‘wildfly-28.0.0.Beta1.zip’

wildfly-28.0.0.Beta1.zip                    100%[=========================================================================================>] 224.58M  62.7MB/s    in 3.6s

2023-04-05 10:13:57 (61.5 MB/s) - ‘wildfly-28.0.0.Beta1.zip’ saved [235488103/235488103]

root@rhel:~# ls
google-cloud.repo  post-run.log  post-run.log.done  rh-cloud.repo  wildfly-28.0.0.Beta1.zip
```

2. Installing “unzip”
$yum install unzip -y

3. Unzipping the wildfly.zip file.
$unzip <filename>

```
root@rhel:~/wildfly-28.0.0.Beta1# ls
appclient  bin  copyright.txt  docs  domain  jboss-modules.jar  LICENSE.txt  modules  README.txt  standalone  welcome-content
root@rhel:~/wildfly-28.0.0.Beta1# cd standalone/
root@rhel:~/wildfly-28.0.0.Beta1/standalone# ls
configuration  deployments  lib  tmp
root@rhel:~/wildfly-28.0.0.Beta1/standalone# cd configuration/
root@rhel:~/wildfly-28.0.0.Beta1/standalone/configuration# ls
application-roles.properties  logging.properties      mgmt-users.properties   standalone-full.xml  standalone-load-balancer.xml    standalone-microprofile.xml
application-users.properties  mgmt-groups.properties  standalone-full-ha.xml  standalone-ha.xml    standalone-microprofile-ha.xml  standalone.xml
root@rhel:~/wildfly-28.0.0.Beta1/standalone/configuration# vi standalone.xml
```

Here,
domain: java can be hosted on 2 servers for high availability.
standalone: java is hosted on only 1 server.
bin : This has many files.

In standalone directory=>configuration directory => standalone.xml file
Changing the public and management IP from localhost “127.0.0.1” to Anywhere “0.0.0.0”

```
<interfaces>
    <interface name="management">
        <inet-address value="${jboss.bind.address.management:127.0.0.1}"/>
    </interface>
    <interface name="public">
        <inet-address value="${jboss.bind.address:127.0.0.1}"/>
    </interface>
</interfaces>
```

Edited file:

```
<interfaces>
    <interface name="management">
        <inet-address value="${jboss.bind.address.management:0.0.0.0}"/>
    </interface>
    <interface name="public">
        <inet-address value="${jboss.bind.address:0.0.0.0}"/>
    </interface>
</interfaces>
```

In bin directory=> running the standalone.sh script

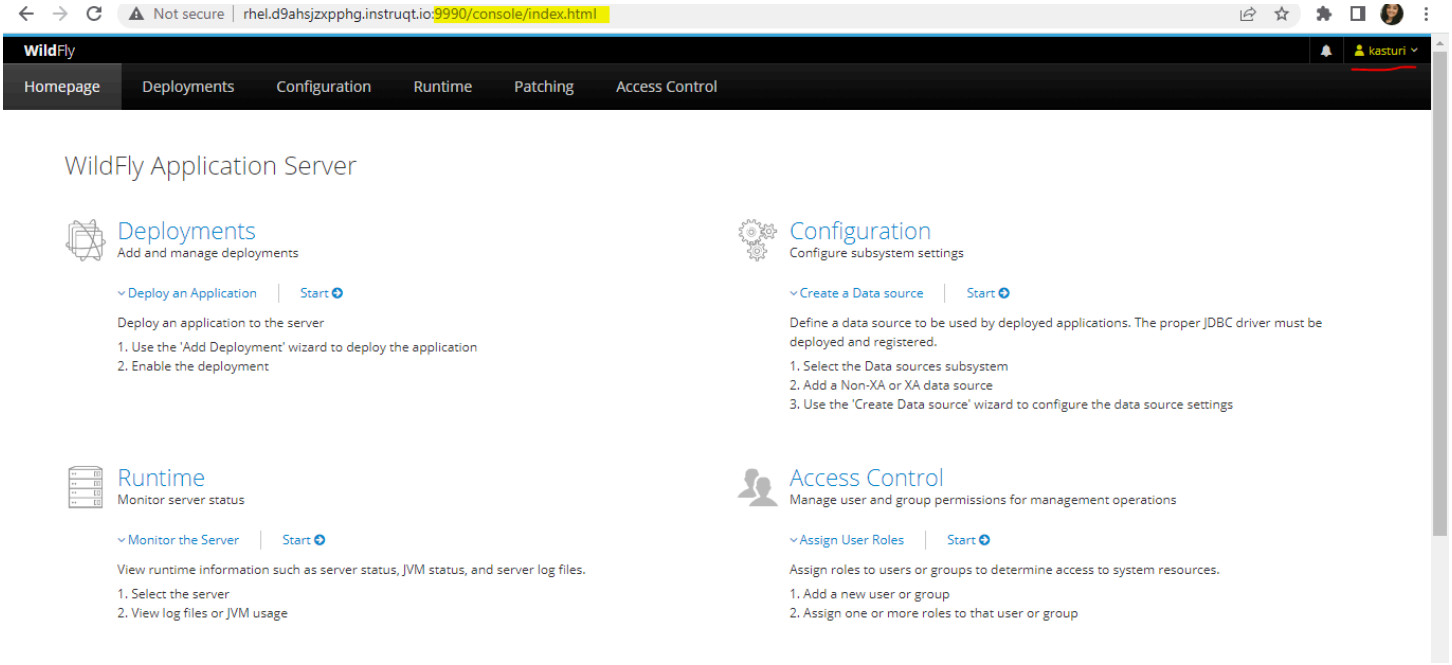
```
root@rhel:~/wildfly-28.0.0.Beta1# ls
appclient  bin  copyright.txt  docs  domain  jboss-modules.jar  LICENSE.txt  modules  README.txt  standalone  welcome-content
root@rhel:~/wildfly-28.0.0.Beta1# cd bin/
root@rhel:~/wildfly-28.0.0.Beta1/bin# ls
add-user.bat         appclient.conf.bat  common.ps1       domain.ps1       jboss-cli-logging.properties  jconsole.sh     standalone.bat       wildfly-elytron-tool.jar  wsprovide.sh
add-user.properties  appclient.conf.ps1  common.sh        domain.sh        jboss-cli.ps1                 jdr.bat         standalone.conf      wsconsume.bat
add-user.ps1         appclient.ps1       domain.bat       elytron-tool.bat jboss-cli.sh                  jdr.ps1         standalone.conf.bat  wsconsume.ps1
add-user.sh          appclient.sh        domain.conf      elytron-tool.ps1 jboss-cli.xml                 jdr.sh          standalone.conf.ps1  wsconsume.sh
appclient.bat        client              domain.conf.bat  elytron-tool.sh  jconsole.bat                  launcher.jar    standalone.ps1       wsprovide.bat
appclient.conf       common.bat          domain.conf.ps1  jboss-cli.bat    jconsole.ps1                  product.conf    standalone.sh        wsprovide.ps1
root@rhel:~/wildfly-28.0.0.Beta1/bin# sh standalone.sh
```

WildFly default page, accessed from port 8080: public end user port

In bin directory => running the add-user.sh script.
We will add a management user “kasturi”

```
root@rhel:~/wildfly-28.0.0.Beta1/bin# sh add-user.sh

What type of user do you wish to add?
 a) Management User (mgmt-users.properties)
 b) Application User (application-users.properties)
(a): a

Enter the details of the new user to add.
Using realm 'ManagementRealm' as discovered from the existing property files.
Username : kasturi
Password recommendations are listed below. To modify these restrictions edit the add-user.properties configuration file.
 - The password should be different from the username
 - The password should not be one of the following restricted values {root, admin, administrator}
 - The password should contain at least 8 characters, 1 alphabetic character(s), 1 digit(s), 1 non-alphanumeric symbol(s)
Password :
WFLYDM0099: Password should have at least 8 characters!
Are you sure you want to use the password entered yes/no? yes
Re-enter Password :
What groups do you want this user to belong to? (Please enter a comma separated list, or leave blank for none)[  ]:
About to add user 'kasturi' for realm 'ManagementRealm'
Is this correct yes/no? yes
Added user 'kasturi' to file '/root/wildfly-28.0.0.Beta1/standalone/configuration/mgmt-users.properties'
Added user 'kasturi' to file '/root/wildfly-28.0.0.Beta1/domain/configuration/mgmt-users.properties'
Added user 'kasturi' with groups  to file '/root/wildfly-28.0.0.Beta1/standalone/configuration/mgmt-groups.properties'
Added user 'kasturi' with groups  to file '/root/wildfly-28.0.0.Beta1/domain/configuration/mgmt-groups.properties'
```

WildFly management console



DOCKERFILE FOR INSTALLING WILDFLY

Dockerfile:

```
FROM docker.io/jboss/base-jdk:11

#Switching to root user, so as to create a dir : mydir
USER root

#creating a directory: mydir
RUN mkdir /mydir

#installing wildfly
ADD https://github.com/wildfly/wildfly/releases/download/28.0.0.Beta1/wildfly-28.0.0.Beta1.zip /mydir/

#unzipping the file: "-d" for specifying the dir, where to unzip.
RUN unzip /mydir/wildfly-28.0.0.Beta1.zip -d /mydir/

#Specifying the work directory:
WORKDIR /mydir/wildfly-28.0.0.Beta1/

#removing the zip file: as we want the container to be lightweight
RUN rm /mydir/wildfly-28.0.0.Beta1.zip

#adding the management user:
RUN sh bin/add-user.sh --silent admin redhat

#running the standalone.sh file and changing the ip access to anywhere => 0.0.0.0
CMD ["./bin/standalone.sh", "-Djboss.bind.address.management=0.0.0.0", "-Djboss.bind.address=0.0.0.0"]
```

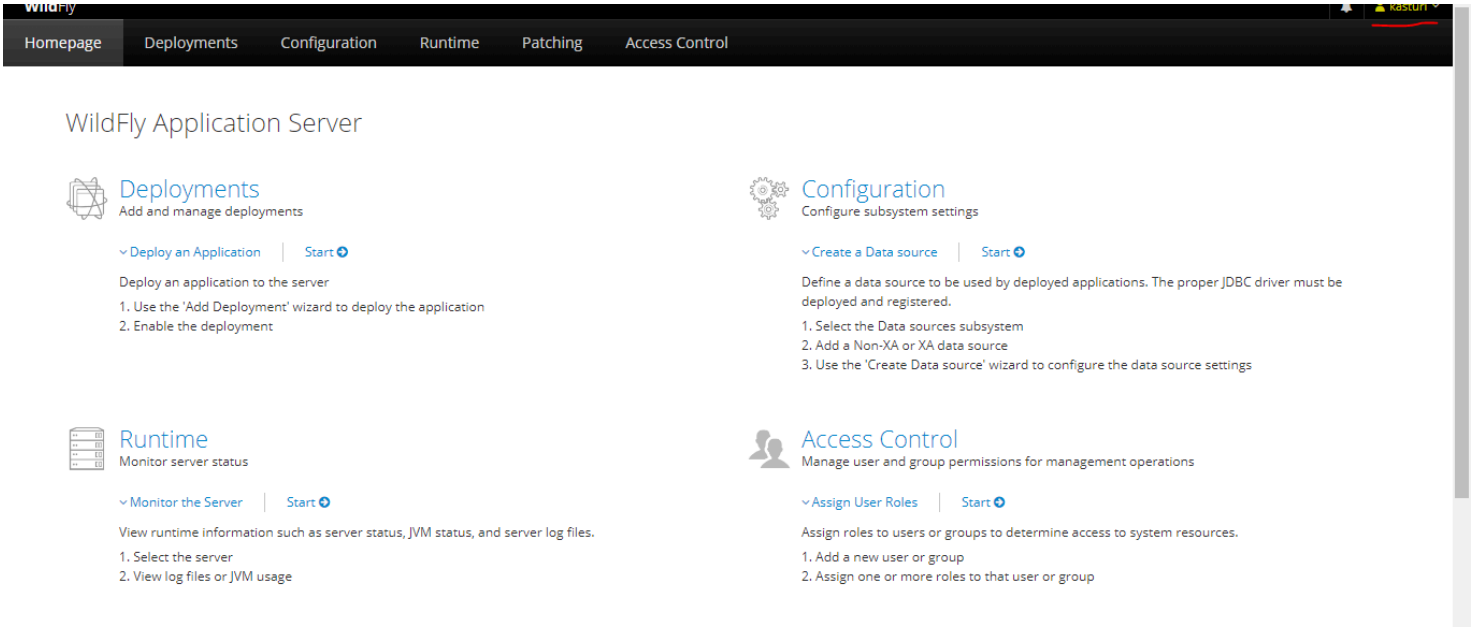
Building the docker image, and creating 2 containers:

1. Public user
2. Management user

```
ubuntu $ docker run -d -p 18080:8080 myimage
94607190c1ec568b9a1149a486c10d0c6c6816931e281fd90070043364c08d3c
ubuntu $ docker run -d -p 28080:9990 myimage
0974868e0e9596a9f6dba6c0e4797103a451fe0ec88b7f7381a3227ae8be3a1c
```

Public WildFly page

Management user's page



DockerFile: ARG Instruction

The ARG instruction in Docker is used to define a build-time variable. These variables can be used during the build process to customise the resulting Docker image.

Example 1:
Creating dockerfile

```
FROM docker.io/ubuntu

#ARG instruction
#here, mydir is the key, and somedir is value of the key which will be assigned during the image build.
#It is completely fine if we don not mention "somedir" here
ARG mydir somedir

WORKDIR $mydir
```

Building the docker file. Giving the value of “mydir”

```
ubuntu $ ls
Dockerfile
ubuntu $ docker build --build-arg mydir=/tmp -t myimg .
```

Running the docker container

```
ubuntu $ docker run -it myimg
root@f4e658999254:/tmp#
```

Example 2:
Writing Dockerfile

```
FROM docker.io/ubuntu

#ARG instruction
#here, src1 is the key, no temporary value is assigned. IT will be assigned during the image build.
ARG src1

COPY $src1 /tmp
```

Here, src1 is the key, it will get its output while image building. It will then copy the specific file mentioned as value to /tmp.

Creating a file “test.sh” on a local machine.

```
ubuntu $ cat test.sh

echo "hello world"
ubuntu $
```

Building the docker img

```
ubuntu $ docker build --build-arg src1=test.sh -t myimg .
```

Here, giving “src1” (key) its value as “test.sh”
Running the docker container

```
ubuntu $ docker run -it myimg
root@70ae9148899c:/# ls /tmp
test.sh
root@70ae9148899c:/# sh /tmp/test.sh
hello world
root@70ae9148899c:/#
```

Dockerfile: HEALTHCHECK - Instruction

Healthcheck tells whether the container is still working or not (actual purpose: to check whether the application inside the container is working or not?)
This can help to prevent issues where a container is running, but not actually serving any traffic.

It can be achieved using 2 methods:
1. Adding healthcheck in Dockerfile itself.
2. Giving the health check parameter while running the docker image.

Method 1: Through Dockerfile

Dockerfile

```
FROM docker.io/busybox

ADD myfile.txt /root/myfile.txt

#instruction: followed by the command
HEALTHCHECK --interval=5s --timeout=3s CMD cat /root/myfile.txt
#this will check whether myfile1.txt is present or not?
#If Present => Health of container= Healthy
#else       => Health of container= Unhealthy

CMD ["ping","google.com"]
```

Building the Docker image.

```
ubuntu $ docker build -t myimg .
Sending build context to Docker daemon  2.048kB
Step 1/4 : FROM docker.io/busybox
latest: Pulling from library/busybox
4b35f584bb4f: Pull complete
Digest: sha256:b5d6fe0712636ceb7430189de28819e195e8966372edfc2d9409d79402a0dc16
Status: Downloaded newer image for busybox:latest
 ---> 7cfbbec8963d
Step 2/4 : ADD myfile.txt /root/myfile.txt
ADD failed: file not found in build context or excluded by .dockerignore: stat myfile.txt: file does not exist
```

This is because there is no file “myfile1.txt” on the host machine.

Let's create a file myfile1.txt and build the docker img AGAIN.

```
ubuntu $ echo "Hello World" > myfile.txt
ubuntu $ docker build -t myimg .
Sending build context to Docker daemon  3.072kB
Step 1/4 : FROM docker.io/busybox
 ---> 7cfbbec8963d
Step 2/4 : ADD myfile.txt /root/myfile.txt
 ---> c69c123bf2f6
Step 3/4 : HEALTHCHECK --interval=5s --timeout=3s CMD cat /root/myfile.txt
 ---> Running in 06e6aed4a095
Removing intermediate container 06e6aed4a095
 ---> 8044b51fc2b9
Step 4/4 : CMD ["ping","google.com"]
 ---> Running in 94b4e5547650
Removing intermediate container 94b4e5547650
 ---> d164ce2fbffd
Successfully built d164ce2fbffd
Successfully tagged myimg:latest
```

Image built successfully.

Now Let's run the container.

```
ubuntu $ docker run -d myimg
5b190d63e28deac49a0f7f464ccd486c3967328b851996bace42a60b113a1e45
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND            CREATED         STATUS                            PORTS     NAMES
5b190d63e28d   myimg     "ping google.com"   2 seconds ago   Up 2 seconds (health: starting)             zealous_colden
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND            CREATED          STATUS                    PORTS     NAMES
5b190d63e28d   myimg     "ping google.com"   23 seconds ago   Up 23 seconds (healthy)             zealous_colden
ubuntu $
```

Note:
We can use : **$watch docker ps**
$watch docker ps is a command used to continuously monitor the running Docker containers in real-time. It executes the docker ps command repeatedly at a set interval, which by default is every two seconds, and displays the output on the terminal.

Method 2: Using - -health-cmd parameter:

`health-cmd` is an option that can be used with the `docker run` command to specify a custom command to perform a health check on a container. This option is used in conjunction with the `--health-start-period` option, which defines the time delay between starting the container and running the health check command.

SYNTAX:
docker run --health-cmd=<command> --health-start-period=<duration> <image>

```
ubuntu $ docker run -d --name=test --health-cmd='stat /etc/nginx/nginx.conf || exit 1' nginx:1.13
06e4989fb7f43bcd7d7ad2e42691cda0fcd3f4e69d7629bb0e792bdb1284d230
```

Stat: checks the status of the file

#watch docker ps

```
Every 2.0s: docker ps                                                                                ubuntu: Mon Apr 17 18:07:25 2023

CONTAINER ID   IMAGE        COMMAND                  CREATED          STATUS                              PORTS     NAMES
06e4989fb7f4   nginx:1.13   "nginx -g 'daemon of   "   10 seconds ago   Up 9 seconds (health: starting)   80/tcp    test
```

Healthy:

```
Every 2.0s: docker ps                                                                                ubuntu: Mon Apr 17 18:08:11 2023

CONTAINER ID   IMAGE        COMMAND                  CREATED          STATUS                          PORTS     NAMES
06e4989fb7f4   nginx:1.13   "nginx -g 'daemon of   "   56 seconds ago   Up 56 seconds (healthy)   80/tcp    test
```

Dockerfile Instruction: ONBUILD:

The ONBUILD instruction is a Dockerfile instruction that adds a **trigger** to the image build process. It allows you to automate certain actions in subsequent builds of an image that are based on the current image.

When you include an ONBUILD instruction in a Dockerfile, the instruction and its arguments are **not executed during the current build process**. Instead, they are executed in a **future build process**, when the **current image** is used as the **base image** for another Dockerfile. This means that the actions specified in the ONBUILD instruction will be performed automatically in the next build of the image.
[Onbuild cmd is executed when the dockerfile is builded into an image, and that image is used in FROM in another Dockerfile.]

NOTE: 1 Layer = 1 instruction

EXAMPLE 1

Dockerfile:

```
FROM docker.io/ubuntu

RUN touch day1.txt

#Onbuild instruction: will be executed in the Second build.
ONBUILD COPY myfile.txt /root/myfile.txt

ONBUILD RUN apt update -y
```

Create myfile.txt on the local machine.
Building the Docker Image **[First build]**

```
ubuntu $ echo "hello" > myfile.txt
ubuntu $ docker build -t myimg .
Sending build context to Docker daemon  3.072kB
Step 1/4 : FROM docker.io/ubuntu
latest: Pulling from library/ubuntu
2ab09b027e7f: Pull complete
Digest: sha256:67211c14fa74f070d27cc59d69a7fa9aeff8e28ea118ef3babc295a0428a6d21
Status: Downloaded newer image for ubuntu:latest
 ---> 08d22c0ceb15
Step 2/4 : RUN touch day1.txt
 ---> Running in d5ed2ad9043c
Removing intermediate container d5ed2ad9043c
 ---> 5fe0f0ebe164
Step 3/4 : ONBUILD COPY myfile.txt /root/myfile.txt
 ---> Running in c09a1787db2d
Removing intermediate container c09a1787db2d
 ---> 970144372521
Step 4/4 : ONBUILD RUN apt update -y
 ---> Running in b93ac1aab2c7
Removing intermediate container b93ac1aab2c7
 ---> 34bf6694217c
Successfully built 34bf6694217c
Successfully tagged myimg:latest
```

$docker inspect myimg

Here, we can see it has created just **2 layers.**

1st layer => first instruction: FROM docker.io/ubuntu

2nd layer=>Second instruction: RUN touch day1.txt

This clearly specifies that the 2 Onbuild commands were not executed, in this build…

```
ubuntu $ docker inspect myimg | grep -A5 Layers
            "Layers": [
                "sha256:b93c1bd012ab8fda60f5b4f5906bf244586e0e3292d84571d3abb56472248466",
                "sha256:eedd6663227ef97a1ecf64628eea2302f792747955b65513f64240d7c8074eaa"
            ]
        },
        "Metadata": {
```

Using this image “myimg” from the First build in a new Dockerfile, and then Building the Docker image **[Second Build]**

```
ubuntu $ ls
Dockerfile  myfile.txt
ubuntu $ vi docf1
ubuntu $ cat docf1
FROM myimg

ubuntu $ docker build -t myimg1 -f docf1 .
```

$docker images

```
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
myimg1       latest    bed41b277736   2 minutes ago    121MB
myimg        latest    34bf6694217c   7 minutes ago    77.8MB
ubuntu       latest    08d22c0ceb15   5 weeks ago      77.8MB
```

Inspecting the new image: myimg1

Here, we have 4 layers

1st layer: FROM docker.io/ubuntu
2nd layer: RUN touch day1.txt
3rd layer: ONBUILD COPY myfile.txt /root/myfile.txt
4th layer: ONBUILD apt update -y

```
ubuntu $ docker inspect myimg1 | grep -A6 Layers
            "Layers": [
                "sha256:b93c1bd012ab8fda60f5b4f5906bf244586e0e3292d84571d3abb56472248466",
                "sha256:eedd6663227ef97a1ecf64628eea2302f792747955b65513f64240d7c8074eaa",
                "sha256:2b52d284270bfee752c57776a61b41018a9967583abb8d18db74e6a77a944610",
                "sha256:1ace203b036a8a37fce007f2b14a4732e892829b0e9a33c48152903407cdc180"
            ]
        },
```

EXAMPLE 2:
Dockerfile:

```
FROM docker.io/ubuntu

RUN apt update -y

RUN apt install apache2 -y

#Onbuild instruction: will be executed in the Second build.
ONBUILD COPY . .

CMD ["/usr/sbin/apache2ctl","-D","FOREGROUND"]
```

Building the Docker image: BUILD 1
$docker build -t image1 .
Pushing the docker image "image1" to docker hub.

```
ubuntu $ docker tag image1 kashdeshmukh/unnatikucl:image1
ubuntu $ docker push kashdeshmukh/unnatikucl:image1
The push refers to repository [docker.io/kashdeshmukh/unnatikucl]
a656a7443e87: Pushed
d77f75a27b0c: Pushed
b93c1bd012ab: Layer already exists
image1: digest: sha256:093d7de1510258227abdde883e199836c69a57fa71a324200848c6b44f452dd1 size: 953
```

Pulling the docker image on a new machine.

```
ubuntu $ docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
ubuntu $ docker pull kashdeshmukh/unnatikucl:image1
image1: Pulling from kashdeshmukh/unnatikucl
2ab09b027e7f: Pull complete
6258f03c2ff3: Pull complete
3746fa638a06: Pull complete
Digest: sha256:093d7de1510258227abdde883e199836c69a57fa71a324200848c6b44f452dd1
Status: Downloaded newer image for kashdeshmukh/unnatikucl:image1
docker.io/kashdeshmukh/unnatikucl:image1
ubuntu $ docker images
REPOSITORY                TAG       IMAGE ID       CREATED         SIZE
kashdeshmukh/unnatikucl   image1    a0fcb90e427f   5 minutes ago   229MB
```

Creating a Dockerfile. This file uses the image pulled as base image:
Building the docker image: Second Build

```
ubuntu $ vi Dockerfile
ubuntu $ cat Dockerfile
FROM kashdeshmukh/unnatikucl:image1


ubuntu $ docker build -t image1 .
Sending build context to Docker daemon  53.02MB
Step 1/1 : FROM kashdeshmukh/unnatikucl:image1
# Executing 1 build trigger
 ---> 42cae5026e0d
Successfully built 42cae5026e0d
Successfully tagged image1:latest
```

Inspecting the docker image: It shows 4 layers.

```
ubuntu $ docker inspect image1 | grep -A5 Layers
            "Layers": [
                "sha256:b93c1bd012ab8fda60f5b4f5906bf244586e0e3292d84571d3abb56472248466",
                "sha256:d77f75a27b0c2b949ee07375eb4742c633a19891f44d33ee9e454416afc970fa",
                "sha256:a656a7443e87b424a1a2004203d6923ad36fe8333b2ddf10276e5d5c08f7c2b0",
                "sha256:d4659e36b811ab8611379065966bd3868383d30637b623315947fc270b42743c"
            ]
```

Limitation of Onbuild:

```
ubuntu $ docker images
REPOSITORY                TAG       IMAGE ID       CREATED          SIZE
image1                    latest    42cae5026e0d   3 minutes ago    281MB
kashdeshmukh/unnatikucl   image1    a0fcb90e427f   12 minutes ago   229MB
```

Whenever we use Onbuild, the size of the image grows dramatically. But the process gets decreased.
Still, this won't help us, as our container needs to be lightweight.

DOCKER FILE OPTIMIZATION

Dockerfile optimization is an important aspect of building efficient and lightweight container images.
Minimise the number of layers: Each instruction in a Dockerfile creates a new layer in the image, so minimising the number of layers can help reduce the size of the image. We can do this by **combining multiple instructions into a single RUN s**tatement and by deleting unnecessary files after each step.

Example:
Dockerfile:

```
#Layer1
FROM docker.io/ubuntu

#multiple run commands
#Layer2
RUN touch day1.txt
#Layer3
RUN mkdir testing
#Layer3
RUN useradd kd
#Layer4
RUN touch myfile.txt

#Layer5
COPY myfile1.txt /root/myfile
```

Building the Docker image, and viewing the Layers created using Inspect.
5 Layers have been created.

```
ubuntu $ docker inspect testing | grep -A5 Layers
            "Layers": [
                "sha256:b93c1bd012ab8fda60f5b4f5906bf244586e0e3292d84571d3abb56472248466",
                "sha256:e3b529f13c2bfdd006a6d1e0f6d3ee56796158f15384ae0717ff3d5877278a4f",
                "sha256:38f06ba4a86860e2e45c80fec2efe738aa28bcde8901f7c5158843b35604505d",
                "sha256:b3409bffe335460fca17b63823db2063880679d1985c1b78fcc5268f88c43678",
                "sha256:550e4bf155721a60bcddb734aca686e0eecafe343b3560dd719b66125c4a3240",
```

Size of the image:

```
ubuntu $ docker images
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
testing             latest    f416a5cbf557   2 minutes ago    78.1MB
```

OPTIMISING THE DOCKER IMAGE:

Dockerfile:
Combining the RUN instructions into one, separated by “&&”

```
#Layer1
FROM docker.io/ubuntu

#multiple run commands
#Layer2
RUN touch day1.txt && mkdir testing && useradd kd && touch myfile.txt

#Layer3
COPY myfile1.txt /root/myfile
```

Building the Docker image;

```
ubuntu $ docker build -t optimizedimg .
Sending build context to Docker daemon  4.096kB
Step 1/3 : FROM docker.io/ubuntu
 ---> 08d22c0ceb15
Step 2/3 : RUN touch day1.txt && mkdir testing && useradd kd && touch myfile.txt
 ---> Running in 0d7fe278cabc
Removing intermediate container 0d7fe278cabc
 ---> 73c460e6ae6c
Step 3/3 : COPY myfile1.txt /root/myfile
 ---> e0b867812252
Successfully built e0b867812252
Successfully tagged optimizedimg:latest
ubuntu $ docker images
REPOSITORY                TAG       IMAGE ID       CREATED          SIZE
optimizedimg              latest    e0b867812252   4 seconds ago    78.1MB
testing                   latest    f416a5cbf557   5 minutes ago    78.1MB
```

Here, the size will remain the same.
Inspecting the Image:
Here, we have just 3 layers.
Thus, Docker file optimization is used to reduce the **"Layers"**

```
ubuntu $ docker inspect optimizedimg | grep -A5 Layers
            "Layers": [
                "sha256:b93c1bd012ab8fda60f5b4f5906bf244586e0e3292d84571d3abb56472248466",
                "sha256:01b2407c7d15dc3b5a73b4b1fc54eb13d66318d1996b18f0e1d8418c986ed176",
                "sha256:daea49fe7c582f466742fb094d1185148c53a4190f03f62bbadd3e2f1ce1c563"
            ]
        },
```

DOCKER MULTI-STAGING BUILD / BUILDER IMAGE

Docker multi-stage builds and builder images are techniques that can help us to create efficient and optimised container images.
Multi-stage builds allow us to use **multiple FROM statements** in a Dockerfile to create different stages of the build process. Each stage can have its own dependencies and tools, and can produce a separate layer in the final image. This allows us to optimise the image size and reduce the attack surface by removing unnecessary files and packages.

- Use multiple FROM statements in 1 dockerfile.
- We only take the **output** from the 1st FROM. And this **won't create the layers of the 1st** FROM.
- The output will be passed on to the next image. [next FROM]
- Thus, **size will be decreased.**

NOTE:
Tollywood ==========> Bollywood
(image1) ===output====>(image1)
1st FROM 2nd FROM
[example in Docker - Tasks document: DOCKER - TASKS]

DEPLOYING MULTI-TIER APPLICATION IN DOCKER

Github link: https://github.com/ashutoshbhakare/kucl2002

Step1: Get the flask application code into a python file**: application.py**
Step 2: Get the requirements into a file: **requirements.txt**

```
ubuntu $ pwd
/root/unnati
ubuntu $ ls
Dockerfile  application.py  requirements.txt
ubuntu $ cat application.py
import time

import redis
from flask import Flask

app = Flask(__name__)
cache = redis.Redis(host='redis', port=6379)

def get_hit_count():
    retries = 5
    while True:
        try:
            return cache.incr('hits')
        except redis.exceptions.ConnectionError as exc:
            if retries == 0:
                raise exc
            retries -= 1
            time.sleep(0.5)

@app.route('/')
def hello():
    count = get_hit_count()
    return 'Hello World! FROM UNNATI I have been seen {} times.\n'.format(count)
ubuntu $ cat requirements.txt
flask
redis
```

Step 3: Write a Dockerfile which will run the flask application: **Dockerfile**

```
#base image
FROM python:3.7-alpine
#changing the work directore (pwd)
WORKDIR /code
#setting up env variable
ENV FLASK_APP=application.py
ENV FLASK_RUN_HOST=0.0.0.0

RUN apk add --no-cache gcc musl-dev linux-headers
#copying the requirements file from local machine to the container.
COPY requirements.txt requirements.txt
#installing all the requirements mwntioned in requirements.txt file.
RUN pip install -r requirements.txt
#exposing the port number
EXPOSE 5000
#copying current files to the container
COPY . .
#running the flask application
CMD ["flask", "run"]
```

Step 4: Building the docker image from Dockerfile:

```
ubuntu $ #docker build -t image1 .
ubuntu $ docker images
REPOSITORY   TAG          IMAGE ID       CREATED          SIZE
image1       latest       64a17a624c81   22 seconds ago   215MB
python       3.7-alpine   c884478a5111   2 weeks ago      47MB
```

Step 5: Creating a network, as it is mandatory for deploying multi-tier applications: **dev**

```
ubuntu $ docker network create dev
d2ddccf981cd4199fccfabf586b97abd132b10be9114ba52de6da400ddaccd01
```

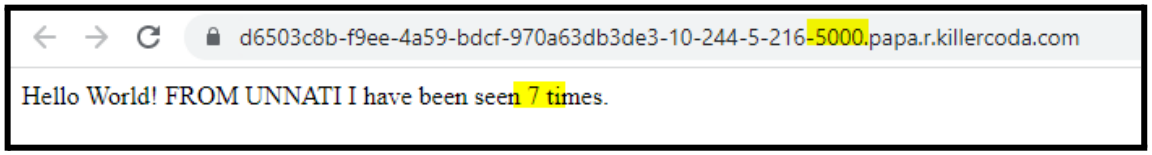
```
ubuntu $ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
d5066c0000ec   bridge    bridge    local
d2ddccf981cd   dev       bridge    local
02789cf894da   host      host      local
9a3042b92bb0   none      null      local
```

Step 6: Creating a container for database: redis [NOSQL Database]: **redis**

```
ubuntu $ #docker run -d --name=redis --network=dev redis
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS      NAMES
e6a8d9eb2abd   redis     "docker-entrypoint.s…"   12 seconds ago   Up 11 seconds   6379/tcp   redis
ubuntu $
```

Step 7: Creating a container for flask application: **flaskapp**

```
ubuntu $ docker run -d -p 5000:5000 --name=flaskapp --network=dev image1
0c4c561b0f4efde6b69d7d92bba1b712cb8bb6efd836425b7ac10e3cb772ee04
ubuntu $ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS              PORTS                                       NAMES
0c4c561b0f4e   image1    "flask run"              3 seconds ago        Up 3 seconds        0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   flaskapp
e6a8d9eb2abd   redis     "docker-entrypoint.s…"   About a minute ago   Up About a minute   6379/tcp                                    redis
ubuntu $
```



DOCKER-COMPOSE: [REFER DOCKER - TASKS]

WHY DO WE NEED DOCKER COMPOSE?

Docker Compose is a tool for defining and running multi-container Docker applications.
It is used to define the services, networks, and volumes required for an application, as well as the relationships between them.
Docker Compose allows us to manage multiple Docker containers as a **single application,** making it easier to deploy and scale our applications.

Therefore**, Advantages of Docker Compose:**

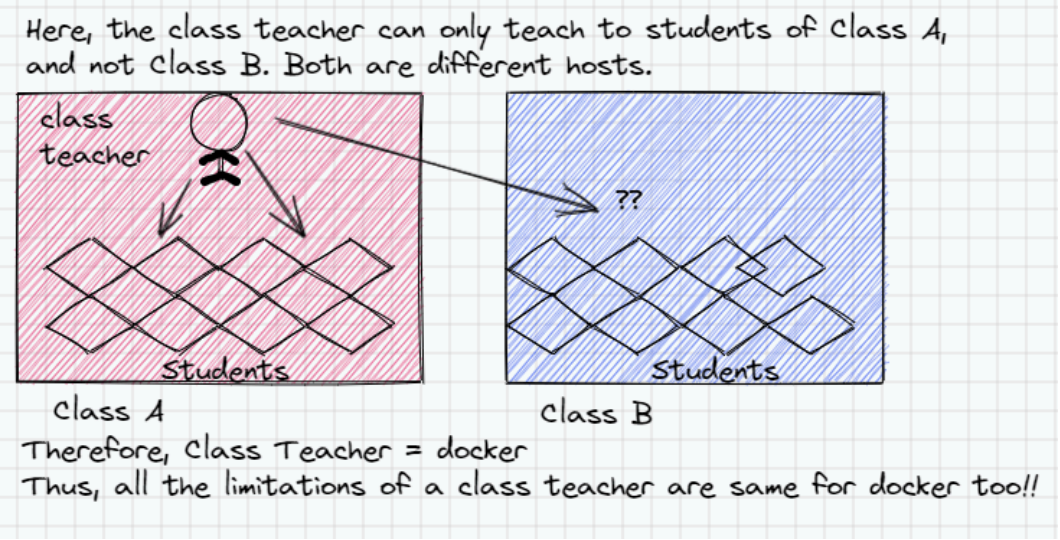
1. **Managing multiple containers**:
Docker Compose allows us to define and manage multiple containers as a single application. This is useful when we need to deploy an application that consists of multiple services, such as a web server, a database, etc.
2. **Easy deployment:**
With Docker Compose, we can define our application in a single YAML file, which makes it easy to deploy our application to different environments, such as development, testing, and production.
3. **Scalability:**
Docker Compose makes it easy to scale our application by adding or removing containers. We can define the number of containers for each service, and Docker Compose will automatically manage the load balancing and scaling for us.

Note: Every Container is denoted as a SERVICE in Docker-Compose.

LIMITATIONS OF DOCKER & ALL THE CONTAINER MANAGEMENT TOOLS

- It is a **SINGLE HOST** container management system. [Single vm, single instance, single machine, etc]
Running many containers on a single host can quickly consume a lot of resources, which can impact the performance of the host.
Therefore, we cannot create containers in multiple hosts.

Example:



- **No High-availability :** This means that in case our container goes down, then we do not have any backup or a second container to cope up with.
- **No Load Balancing,** as no replicas are present.
- **Security** : Containers share the same kernel as the host, which means that a vulnerability in the kernel can potentially affect all containers running on that host.
- **No Centralised management.**

RACE CONDITION

Race conditions can occur in Docker (or anywhere else) when multiple processes or containers or servers or replicas or node clusters are accessing shared resources, such as the same file or database, simultaneously. For example, if two containers are trying to write to the same file at the same time, a race condition can occur where one container overwrites the other's changes or the file becomes corrupt.
OR
One container will say that the 2nd container will do the work, and vice versa. In this case, the work won't be completed, and the end user won't be able to access the final application.

Here, there is **NO MANAGER to manage or centralise the system.**
Therefore,
To prevent race conditions in Docker, it's important to use **container orchestration tools** like **Kubernetes or Docker Swarm [HOD]**, which provide mechanisms for managing and synchronising multiple containers.

ORCHESTRATION

Orchestration refers to the process of managing and coordinating multiple containers running in a distributed environment. Container orchestration involves automating the deployment, scaling, and management of containers, as well as ensuring that they are running correctly and communicating with each other effectively.
Orchestration tools like Docker Swarm and Kubernetes provide a way to manage and automate the deployment and management of containers across a cluster of machines. These tools offer features like load balancing, service discovery, health checking, and automatic scaling, which help ensure that containerized applications are running smoothly and efficiently.

Thus, Orchestration is required for:
- Load Balancing
- High Availability
- Centralised Management

Note:
Until now, we studied **Docker CE = Docker Community Edition**. It is Open Source.
Docker EE = Docker Enterprise Edition. => Mirantis
**Swarm is not Enterprise grade.

Products of Mirantis:

Mirantis Kubernetes Engine

Simple, flexible, and scalable container orchestration.

Container orchestration is powerful—but no one ever said it was easy. Avoid the pain and get back to mission-critical engineering with Mirantis Kubernetes Engine, the enterprise-ready platform for deploying containers at scale.

Mirantis Container Runtime

Secure, industry-standard container runtime—Docker interface included.

Enterprises shouldn't have to worry about container runtime security. That's why we provide the same workhorse runtime at the heart of our ZeroOps stack on a standalone basis.

Mirantis Container Runtime provides the keystone of a secure software supply chain for organizations that need an enterprise-grade container engine to build and run mission-critical containerized workloads.

Mirantis Secure Registry

Your private, cloud native hub for container images.

Software supply chain attacks paralyze businesses—often introducing malicious container images early in the development pipeline. Public registries are rife with corrupted images. How can you keep development moving forward swiftly and securely?

Mirantis Secure Registry provides an enterprise grade container registry solution that can be easily integrated to provide the core of an effective secure software supply chain.

DOCKER SWARM

ORCHESTRATION

Orchestration refers to the process of managing and coordinating multiple containers running in a distributed environment.
Container orchestration involves automating the deployment, scaling, and management of containers, as well as ensuring that they are running correctly and communicating with each other effectively.

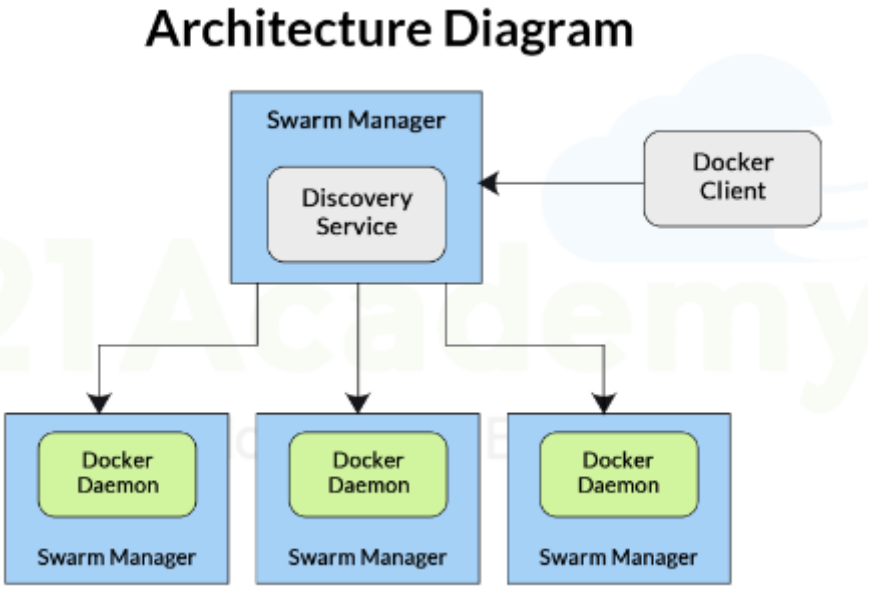
Orchestration tools like **Docker Swarm and Kubernetes** provide a way to manage and automate the deployment and management of containers across a cluster of machines.

DOCKER SWARM

Docker Swarm is a container orchestration tool used to manage and deploy Docker containers across multiple hosts. It allows users to create a cluster of Docker nodes that work together to provide a highly available, scalable, and fault-tolerant platform for running containerized applications.

DOCKER SWARM ARCHITECTURE

1. Swarm Manager / Master / Control Plane
2. Swarm Nodes / Worker Nodes [They create containers]



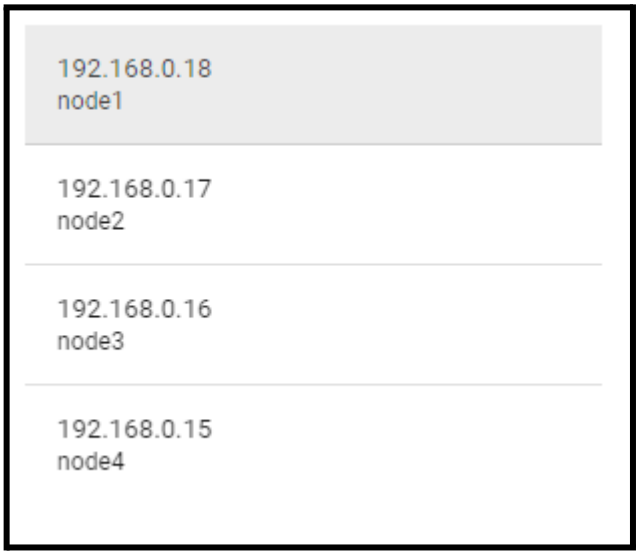
LIMITATION OF SWARM:

Swarm manager also creates the container.

PREREQUISITE:

1. All manager and nodes should be in the same network.
2. All must have Docker installed on them.
3. We need the IP address of the manager.
4. We need open ports between hosts.

Docker nodes



Node1 = Master node
Node2, Node3, Node4 = Worker nodes

Creating Node 1 as Master Node:

```
$ docker swarm init --advertise-addr="192.168.0.18"
Swarm initialized: current node (n1sclaltxpxiu7va5x265tqt5) is now a manage
r.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-01k1kdfwpqow363ywxdysdkt04oq5vjjozty
rqhurhwsrhwl1t-6sysfe9ytp31rf7khb9cz8au3 192.168.0.18:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and f
ollow the instructions.
```

Creating Node2 as Worker node 1:

```
[node2] (local) root@192.168.0.17 ~
$ docker swarm join --token SWMTKN-1-01k1kdfwpqow363ywxdysdkt04oq5vjjoztyrq
hurhwsrhwl1t-6sysfe9ytp31rf7khb9cz8au3 192.168.0.18:2377
This node joined a swarm as a worker.
[node2] (local) root@192.168.0.17 ~
$
```

Creating Node3 as Worker node 1:

```
[node3] (local) root@192.168.0.16 ~
$ docker swarm join --token SWMTKN-1-01k1kdfwpqow363ywxdysdkt04oq5vjjoztyrq
hurhwsrhwl1t-6sysfe9ytp31rf7khb9cz8au3 192.168.0.18:2377
This node joined a swarm as a worker.
```

Creating Node4 as Worker node 1:

```
[node4] (local) root@192.168.0.15 ~
$ docker swarm join --token SWMTKN-1-01k1kdfwpqow363ywxdysdkt04oq5vjjoztyrq
hurhwsrhwl1t-6sysfe9ytp31rf7khb9cz8au3 192.168.0.18:2377
This node joined a swarm as a worker.
```

On Master Node: Node1

Listing the nodes:

```
[node1] (local) root@192.168.0.18 ~
$ docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER S
TATUS   ENGINE VERSION
n1sclaltxpxiu7va5x265tqt5 *   node1      Ready     Active         Leader
        20.10.17
bfrvy0j70kl5djo1uy6uemx4h     node2      Ready     Active
        20.10.17
ysqma1wmv3rzgbtzvjlfm4uoj     node3      Ready     Active
        20.10.17
m13qudh5c5bundrhv5f1m8vn5     node4      Ready     Active
        20.10.17
```

In Docker Swarm, Service == Container.
Thus, creating a container “test”, with 5 replicas / copies of it.

```
[node1] (local) root@192.168.0.18 ~
$ docker service create --name test --replicas=5 httpd
i389b91av8wn1nixfd00vg00i
overall progress: 5 out of 5 tasks
1/5: running
2/5: running
3/5: running
4/5: running
5/5: running
verify: Service converged
```

Listing the Docker service:

```
[node1] (local) root@192.168.0.18 ~
$ docker service ls
ID             NAME      MODE         REPLICAS   IMAGE          PORTS
i389b91av8wn   test      replicated   5/5        httpd:latest
```

Listing the replicas in the service

```
$ docker service ps test
ID             NAME      IMAGE          NODE      DESIRED STATE   CURRENT S
TATE               ERROR     PORTS
rqa93t653ovn   test.1    httpd:latest   node2     Running         Running a
bout a minute ago
w8vcvpg292rc   test.2    httpd:latest   node3     Running         Running a
bout a minute ago
b3guvhcfu4qt   test.3    httpd:latest   node4     Running         Running a
bout a minute ago
vlpyx7ohtvja   test.4    httpd:latest   node1     Running         Running a
bout a minute ago
b9657dldioe0   test.5    httpd:latest   node1     Running         Running a
bout a minute ago
```

Here, each replica is automatically associated with a different worker node.
What if we delete any node from the worker node:
Let’s delete node4:

Now, the service replica of node4 is shifted to node2 “automatically”.

```
[node1] (local) root@192.168.0.18 ~
$ docker service ps test
ID             NAME          IMAGE          NODE      DESIRED STATE   CURREN
T STATE           ERROR      PORTS
rqa93t653ovn   test.1        httpd:latest   node2     Running         Runnin
g 2 minutes ago
w8vcvpg292rc   test.2        httpd:latest   node3     Running         Runnin
g 2 minutes ago
m4gqa8c05lxf   test.3        httpd:latest   node2     Ready           Ready
1 second ago
b3guvhcfu4qt    \_ test.3    httpd:latest   node4     Shutdown        Runnin
g 2 minutes ago
vlpyx7ohtvja   test.4        httpd:latest   node1     Running         Runnin
g 2 minutes ago
b9657dldioe0   test.5        httpd:latest   node1     Running         Runnin
g 2 minutes ago
```

Let’s delete node3:
Service on node3 shifted to node1

```
$ docker service ps test
ID             NAME          IMAGE          NODE      DESIRED STATE   CURREN
T STATE                ERROR      PORTS
rqa93t653ovn   test.1        httpd:latest   node2     Running         Runnin
g 4 minutes ago
smy9wudbymiq   test.2        httpd:latest   node1     Ready           Ready
2 seconds ago
w8vcvpg292rc    \_ test.2    httpd:latest   node3     Shutdown        Runnin
g 4 minutes ago
m4gqa8c05lxf   test.3        httpd:latest   node2     Running         Runnin
g about a minute ago
b3guvhcfu4qt    \_ test.3    httpd:latest   node4     Shutdown        Runnin
g 4 minutes ago
vlpyx7ohtvja   test.4        httpd:latest   node1     Running         Runnin
g 4 minutes ago
b9657dldioe0   test.5        httpd:latest   node1     Running         Runnin
g 4 minutes ago
```

Let’s delete node2:
Now, every service is solely running on Node1 i.e. the master node.

```
$ docker service ps test
ID             NAME          IMAGE          NODE      DESIRED STATE   CURREN
T STATE            ERROR      PORTS
70jn52u92idj   test.1        httpd:latest   node1     Ready           Ready
1 second ago
rqa93t653ovn    \_ test.1    httpd:latest   node2     Shutdown        Runnin
g 5 minutes ago
smy9wudbymiq   test.2        httpd:latest   node1     Running         Runnin
g 50 seconds ago
w8vcvpg292rc    \_ test.2    httpd:latest   node3     Shutdown        Runnin
g 5 minutes ago
qd5sls38p1wj   test.3        httpd:latest   node1     Ready           Ready
1 second ago
m4gqa8c05lxf    \_ test.3    httpd:latest   node2     Shutdown        Runnin
g 2 minutes ago
b3guvhcfu4qt    \_ test.3    httpd:latest   node4     Shutdown        Runnin
g 5 minutes ago
```

This is what the benefit of orchestration is. No manual work needed!!

TASK:
We create swarm worker nodes and manager nodes using the token that appears when we create a manager node.
But what if we want to create the other nodes after some time, and then we don;t have the key?
How to bring the key / token back??

Firstly, creating a Docker Swarm Manager / Master. [Node1]

When we create a docker master, we get a **token value** through which we can create the worker nodes.
But what if at some point, in the future, we want to create / add a worker node to the swarm? And that time we don't have the token?
$docker swarm --help

```
$ docker swarm --help

Usage:  docker swarm COMMAND

Manage Swarm

Commands:
  ca          Display and rotate the root CA
  init        Initialize a swarm
  join        Join a swarm as a node and/or manager
  join-token  Manage join tokens
  leave       Leave the swarm
  unlock      Unlock swarm
  unlock-key  Manage the unlock key
  update      Update the swarm
```

Let's see the syntax of join-token:

```
$ docker swarm join-token --help

Usage:  docker swarm join-token [OPTIONS] (worker|manager)
```

Thus, using this we can create tokens for either "worker" or "manager"

Let's view the **Worker Token:**

```
[node1] (local) root@192.168.0.13 ~
$ docker swarm join-token worker
To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-0sh8hwn0i7sd0q540ylwyet5fn5iry9wskasao0gsuerooyz5d-4yknzo692rmd
klcdd5oz1h1lt 192.168.0.13:2377
```

Creating a worker node using this token: [Node2]

```
[node2] (local) root@192.168.0.12 ~
$ddocker swarm join --token SWMTKN-1-0sh8hwn0i7sd0q540ylwyet5fn5iry9wskasao0gsuerooyz5d-4yknz
o692rmdklcdd5oz1h1lt 192.168.0.13:2377
This node joined a swarm as a worker.
```

On Node1 = Manager:

```
[node1] (local) root@192.168.0.13 ~
$ docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
fvdtbjka4x5i24p3tye4j9jw3 *   node1      Ready     Active         Leader           20.10.17
3fkiug4z29lzu7i2wzw308osi     node2      Ready     Active                          20.10.17
```

Created a node2:

On node1, created a token for manager:

```
[node1] (local) root@192.168.0.13 ~
$ docker swarm join-token manager
To add a manager to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-0sh8hwn0i7sd0q540ylwyet5fn5iry9wskasao0gsuerooyz5d-7phqlup1ij3p
961wpg2wx8le4 192.168.0.13:2377
```

Converting Node3 as a Manager. [Secondary manager]

```
[node3] (local) root@192.168.0.11 ~
$ docker swarm join --token SWMTKN-1-0sh8hwn0i7sd0q540ylwyet5fn5iry9wskasao0gsuerooyz5d-7phql
up1ij3p961wpg2wx8le4 192.168.0.13:2377
This node joined a swarm as a manager.
```

On Node1:

```
[node1] (local) root@192.168.0.13 ~
$ docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
fvdtbjka4x5i24p3tye4j9jw3 *   node1      Ready     Active         Leader           20.10.17
3fkiug4z29lzu7i2wzw308osi     node2      Ready     Active                          20.10.17
zgb45mmdzfxn9kpkxsk70t6n3     node3      Ready     Active         Reachable        20.10.17
```

DOCKER SWARM CONTINUED

On Node1: Created it as a **Manager / Master.**
Trying to Leave the swarm

```
[node1] (local) root@192.168.0.13 ~
$ docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
fvdtbjka4x5i24p3tye4j9jw3 *   node1      Ready     Active         Leader           20.10.17
3fkiug4z29lzu7i2wzw308osi     node2      Ready     Active                          20.10.17
zgb45mmdzfxn9kpkxsk70t6n3     node3      Ready     Active         Reachable        20.10.17
[node1] (local) root@192.168.0.13 ~
$ docker swarm leave
Error response from daemon: You are attempting to leave the swarm on a node that is participating as a
 manager. Removing this node leaves 1 managers out of 2. Without a Raft quorum your swarm will be inac
cessible. The only way to restore a swarm that has lost consensus is to reinitialize it with `--force-
new-cluster`. Use `--force` to suppress this message.
```

Using the –force option:

```
[node1] (local) root@192.168.0.13 ~
$ docker swarm leave --force
Node left the swarm.
[node1] (local) root@192.168.0.13 ~
$ docker node ls
Error response from daemon: This node is not a swarm manager. Use "docker swarm init" or "docker swarm
 join" to connect this node to swarm and try again.
```

Thus, the Node1 as in the Manager left the swarm.

Again, adding Node1 as the manager of the swarm cluster:

```
[node1] (local) root@192.168.0.13 ~
$ docker swarm init --advertise-addr 192.168.0.13
Swarm initialized: current node (lw4122k3hoydbl76b440jbccr) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-3jzm0c8jz78ubh7a64rlyzoqyy3yku45h0cg2nfbz00n4wmcl8-2zqljcexby2m
vjnxgi738t9sj 192.168.0.13:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

Docker Swarm Service Parameters

```
$ docker service --help

Usage:  docker service COMMAND

Manage services

Commands:
  create      Create a new service
  inspect     Display detailed information on one or more services
  logs        Fetch the logs of a service or task
  ls          List services
  ps          List the tasks of one or more services
  rm          Remove one or more services
  rollback    Revert changes to a service's configuration
  scale       Scale one or multiple replicated services
  update      Update a service
```

1. **CREATE: creates the service**

```
$ docker service create -p 18080:80 --name test --replicas=4 docker.io/httpd
t8q0tirz9irhrqiy5bjqg2fjj
overall progress: 4 out of 4 tasks
1/4: running   [==================================================>]
2/4: running   [==================================================>]
3/4: running   [==================================================>]
4/4: running   [==================================================>]
verify: Service converged
```

2. **INSPECT: inspects the service**

```
$ docker inspect test
[
    {
        "ID": "t8q0tirz9irhrqiy5bjqg2fjj",
        "Version": {
            "Index": 13
        },
        "CreatedAt": "2023-05-01T06:25:55.711095947Z",
        "UpdatedAt": "2023-05-01T06:25:55.712735249Z",
        "Spec": {
            "Name": "test",
            "Labels": {},
            "TaskTemplate": {
                "ContainerSpec": {
                    "Image": "httpd:latest@sha256:a182ef2350699f04b
```

3. **LOGS:**

```
$ docker service logs test
test.3.l32gwd34brb4@node1    | AH00557: httpd: apr sockaddr info get() failed for 34214b7e1bee
test.3.l32gwd34brb4@node1    | AH00558: httpd: Could not reliably determine the server's fully qualifi
ed domain name, using 127.0.0.1. Set the 'ServerName' directive globally to suppress this message
test.3.l32gwd34brb4@node1    | AH00557: httpd: apr sockaddr info get() failed for 34214b7e1bee
test.1.c2iy2w2lybl7@node1    | AH00557: httpd: apr sockaddr info get() failed for 455eb0d0ddc8
test.3.l32gwd34brb4@node1    | AH00558: httpd: Could not reliably determine the server's fully qualifi
ed domain name, using 127.0.0.1. Set the 'ServerName' directive globally to suppress this message
```

4. **LS : List the services:**

```
$ docker service ls
ID             NAME      MODE         REPLICAS   IMAGE          PORTS
t8q0tirz9irh   test      replicated   4/4        httpd:latest   *:18080->80/tcp
```

5. **PS : List the components: containers / replicas inside the service**

```
$ docker service ps test
ID             NAME      IMAGE          NODE      DESIRED STATE   CURRENT STATE           ERROR     PO
RTS
c2iy2w2lybl7   test.1    httpd:latest   node1     Running         Running 6 minutes ago
8yll64vncj4x   test.2    httpd:latest   node1     Running         Running 6 minutes ago
l32gwd34brb4   test.3    httpd:latest   node1     Running         Running 6 minutes ago
zf491h551av1   test.4    httpd:latest   node1     Running         Running 6 minutes ago
```

6. **UPDATE: updates a service**

Updating the docker image from httpd:latest to httpd:2

```
$ docker service update --image=httpd:2 test
test
overall progress: 4 out of 4 tasks
1/4: running   [==================================================>]
2/4: running   [==================================================>]
3/4: running   [==================================================>]
4/4: running   [==================================================>]
verify: Service converged
[node1] (local) root@192.168.0.13 ~
$ docker service ls
ID             NAME      MODE         REPLICAS   IMAGE      PORTS
t8q0tirz9irh   test      replicated   4/4        httpd:2    *:18080->80/tcp
```

7. **ROLLBACK: reverts the changes (undo)**

```
$ docker service rollback test
test
rollback: manually requested rollback
overall progress: rolling back update: 4 out of 4 tasks
1/4: running   [>                                                  ]
2/4: running   [>                                                  ]
3/4: running   [>                                                  ]
4/4: running   [>                                                  ]
verify: Service converged
[node1] (local) root@192.168.0.13 ~
$ docker service ls
ID             NAME      MODE         REPLICAS   IMAGE          PORTS
t8q0tirz9irh   test      replicated   4/4        httpd:latest   *:18080->80/tcp
```

It creates a new container whenever any change is made.

```
$ docker service ps test
ID             NAME         IMAGE          NODE      DESIRED STATE   CURRENT STATE            ERROR     PORTS
455svsqukl5a   test.1       httpd:latest   node1     Running         Running 4 minutes ago
zvx9zwji91dk    \_ test.1   httpd:2        node1     Shutdown        Shutdown 4 minutes ago
c2iy2w2lybl7    \_ test.1   httpd:latest   node1     Shutdown        Shutdown 6 minutes ago
69hovr56g436   test.2       httpd:latest   node1     Running         Running 4 minutes ago
18ksh7g4lbkf    \_ test.2   httpd:2        node1     Shutdown        Shutdown 4 minutes ago
8yll64vncj4x    \_ test.2   httpd:latest   node1     Shutdown        Shutdown 6 minutes ago
0pzxvr4engts   test.3       httpd:latest   node1     Running         Running 4 minutes ago
33sabvv3a3oa    \_ test.3   httpd:2        node1     Shutdown        Shutdown 4 minutes ago
132gwd34brb4    \_ test.3   httpd:latest   node1     Shutdown        Shutdown 6 minutes ago
kvhcw3tyj3p8   test.4       httpd:latest   node1     Running         Running 4 minutes ago
8bhpjxsw9stt    \_ test.4   httpd:2        node1     Shutdown        Shutdown 4 minutes ago
zf491h551av1    \_ test.4   httpd:latest   node1     Shutdown        Shutdown 6 minutes ago
```

8. **SCALE:**

Here, we have 4 replicas. Let;s scale them up to 6:

```
$ docker service scale test=6
test scaled to 6
overall progress: 6 out of 6 tasks
1/6: running   [==================================================>]
2/6: running   [==================================================>]
3/6: running   [==================================================>]
4/6: running   [==================================================>]
5/6: running   [==================================================>]
6/6: running   [==================================================>]
verify: Service converged
```

Verify:

```
$ docker service ps test
ID             NAME         IMAGE          NODE      DESIRED STATE   CURRENT STATE             ERROR     PORTS
455svsqukl5a   test.1       httpd:latest   node1     Running         Running 10 minutes ago
zvx9zwji91dk    \_ test.1   httpd:2        node1     Shutdown        Shutdown 10 minutes ago
c2iy2w2lybl7    \_ test.1   httpd:latest   node1     Shutdown        Shutdown 13 minutes ago
69hovr56g436   test.2       httpd:latest   node1     Running         Running 10 minutes ago
18ksh7g4lbkf    \_ test.2   httpd:2        node1     Shutdown        Shutdown 10 minutes ago
8yll64vncj4x    \_ test.2   httpd:latest   node1     Shutdown        Shutdown 13 minutes ago
0pzxvr4engts   test.3       httpd:latest   node1     Running         Running 10 minutes ago
33sabvv3a3oa    \_ test.3   httpd:2        node1     Shutdown        Shutdown 10 minutes ago
132gwd34brb4    \_ test.3   httpd:latest   node1     Shutdown        Shutdown 12 minutes ago
kvhcw3tyj3p8   test.4       httpd:latest   node1     Running         Running 10 minutes ago
8bhpjxsw9stt    \_ test.4   httpd:2        node1     Shutdown        Shutdown 10 minutes ago
zf491h551av1    \_ test.4   httpd:latest   node1     Shutdown        Shutdown 13 minutes ago
3djwrfh22uld   test.5       httpd:latest   node1     Running         Running 2 minutes ago
jr2c3abwed09   test.6       httpd:latest   node1     Running         Running 2 minutes ago
```

9. **RM : removes the service:**

Creating a service “test” with 10 container replicas:

```
$ docker service create --name test --replicas=10 docker.io/httpd
55vu666fml091u8em66p7eqk4
overall progress: 10 out of 10 tasks
1/10: running   [==================================================>]
2/10: running   [==================================================>]
3/10: running   [==================================================>]
4/10: running   [==================================================>]
5/10: running   [==================================================>]
6/10: running   [==================================================>]
7/10: running   [==================================================>]
8/10: running   [==================================================>]
9/10: running   [==================================================>]
10/10: running   [==================================================>]
verify: Service converged
```

Listing:

```
$ docker service ls
ID             NAME      MODE         REPLICAS   IMAGE          PORTS
55vu666fml09   test      replicated   10/10      httpd:latest
[node1] (local) root@192.168.0.13 ~
$ docker service ps test
ID             NAME       IMAGE          NODE      DESIRED STATE   CURRENT STATE            ERROR     PORTS
oanjxmux0gt3   test.1     httpd:latest   node1     Running         Running 55 seconds ago
vfl0dmxynzam   test.2     httpd:latest   node1     Running         Running 54 seconds ago
y9s5qp76zcbv   test.3     httpd:latest   node1     Running         Running 54 seconds ago
sq42ni7d0ndp   test.4     httpd:latest   node1     Running         Running 54 seconds ago
ee984hejnxsa   test.5     httpd:latest   node1     Running         Running 54 seconds ago
hwjrsmksv78w   test.6     httpd:latest   node1     Running         Running 54 seconds ago
mb17x3t68tyr   test.7     httpd:latest   node1     Running         Running 54 seconds ago
jlgj9qeaebku   test.8     httpd:latest   node1     Running         Running 55 seconds ago
ai7romop48fb   test.9     httpd:latest   node1     Running         Running 54 seconds ago
8xl30u2zttp2   test.10    httpd:latest   node1     Running         Running 54 seconds ago
```

Removing the Containers in the Service:

```
$ docker rm -f $(docker ps -q)
8b49b5c6d5fc
2525d936a45f
7b4f49e301ad
48931af5ab0c
bf95a793bb29
bbc811f60fc2
0ca385ee5fe6
1e02aa112985
6d690134c952
e251574140dc
```

Listing:

```
$ docker service ps test
ID             NAME        IMAGE          NODE      DESIRED STATE   CURRENT STATE            ERROR                         PORTS
j3eifq8gd377   test.1      httpd:latest   node1     Running         Running 52 seconds ago
oanjxmux0gt3    \_ test.1  httpd:latest   node1     Shutdown        Failed 58 seconds ago    "task: non-zero exit (137)"
4ui8m9fa2ziu   test.2      httpd:latest   node1     Running         Running 52 seconds ago
vfl0dmxynzam    \_ test.2  httpd:latest   node1     Shutdown        Failed 58 seconds ago    "task: non-zero exit (137)"
8r3uhseksk fy  test.3      httpd:latest   node1     Running         Running 52 seconds ago
y9s5qp76zcbv    \_ test.3  httpd:latest   node1     Shutdown        Failed 58 seconds ago    "task: non-zero exit (137)"
j6aq7sbduoy4   test.4      httpd:latest   node1     Running         Running 52 seconds ago
sq42ni7d0ndp    \_ test.4  httpd:latest   node1     Shutdown        Failed 58 seconds ago    "task: non-zero exit (137)"
yi6q8c4dp69u   test.5      httpd:latest   node1     Running         Running 52 seconds ago
ee984hejnxsa    \_ test.5  httpd:latest   node1     Shutdown        Failed 58 seconds ago    "task: non-zero exit (137)"
usf7b81a64fe   test.6      httpd:latest   node1     Running         Running 53 seconds ago
hwjrsmksv78w    \_ test.6  httpd:latest   node1     Shutdown        Failed 58 seconds ago    "task: non-zero exit (137)"
3hu984mhmnbf   test.7      httpd:latest   node1     Running         Running 52 seconds ago
mb17x3t68tyr    \_ test.7  httpd:latest   node1     Shutdown        Failed 58 seconds ago    "task: non-zero exit (137)"
s4wh9hq1743h   test.8      httpd:latest   node1     Running         Running 53 seconds ago
jlgj9qeaebku    \_ test.8  httpd:latest   node1     Shutdown        Failed 58 seconds ago    "task: non-zero exit (137)"
miiiaozfhmid   test.9      httpd:latest   node1     Running         Running 53 seconds ago
ai7romop48fb    \_ test.9  httpd:latest   node1     Shutdown        Failed 59 seconds ago    "task: non-zero exit (137)"
2fplwku9aq9x   test.10     httpd:latest   node1     Running         Running 52 seconds ago
8xl30u2zttp2    \_ test.10 httpd:latest   node1     Shutdown        Failed 58 seconds ago    "task: non-zero exit (137)"
```

Even if we delete all the container replicas in the service, it will automatically create new container replicas.

But, if we delete the Service, then it will delete the entire containers, and remove them permanently.

```
$ docker service rm test
test
```

```
$ docker service ls
ID          NAME      MODE      REPLICAS    IMAGE      PORTS
```

Finally deleted!!

Example by ASHUTOSH S. BHAKARE SIR

The best example for understanding this is as follows:

“Aasli Raavan voh hai Jiska saaya hai!!”

Therefore, killing the Raavan with a shadow will kill all its replicas.

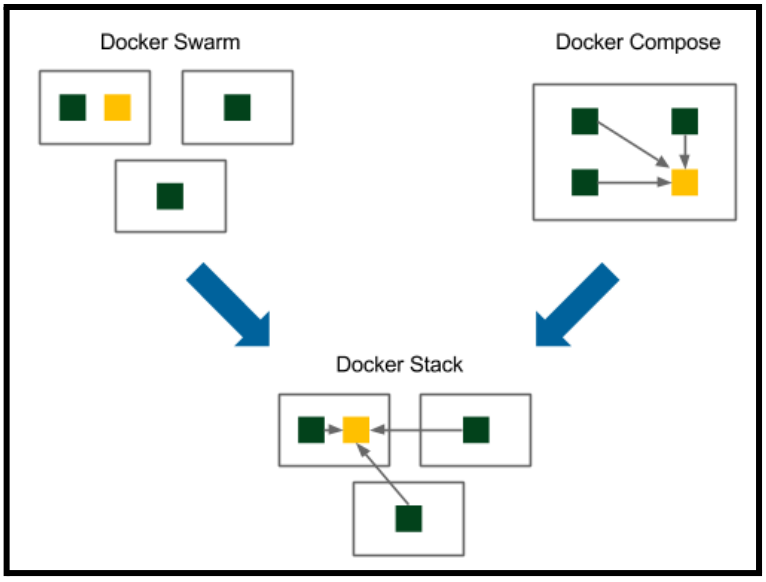
In the same way, removing the **service** will remove all the **container** replicas.

DOCKER STACK

Docker Stack = Docker Compose + Docker Swarm

Docker Compose = Single Node
Docker Swarm = Multi-Node
Therefore, Docker Stack = Multi Container Management system

Docker Stack takes a compose file & creates containers in “Multi-host” or “Multi-Node” environments.
Even though all containers are in different nodes, still they will communicate with each other, as they are in the same cluster.



Deploying a Docker Stack Application:

Link: https://github.com/dockersamples/example-voting-app

```
$ git clone https://github.com/dockersamples/example-voting-app
Cloning into 'example-voting-app'...
remote: Enumerating objects: 1084, done.
remote: Total 1084 (delta 0), reused 0 (delta 0), pack-reused 1084
Receiving objects: 100% (1084/1084), 1.14 MiB | 16.90 MiB/s, done.
Resolving deltas: 100% (408/408), done.
[node1] (local) root@192.168.0.28 ~
$ ls
example-voting-app
[node1] (local) root@192.168.0.28 ~
$ cd example-voting-app/
[node1] (local) root@192.168.0.28 ~/example-voting-app
$ ls
LICENSE                     docker-compose.yml          seed-data
MAINTAINERS                 docker-stack.yml            vote
README.md                   healthchecks                worker
architecture.excalidraw.png k8s-specifications
docker-compose.images.yml   result
[node1] (local) root@192.168.0.28 ~/example-voting-app
```

Syntax of Docker Stack:
$docker stack [OPTIONS] COMMAND

Creating the Node1 as Manager / Master Node:

```
[node1] (local) root@192.168.0.28 ~/example-voting-app
$ docker swarm init --advertise-addr 192.168.0.28
Swarm initialized: current node (vnfp50jko9ao7xl76u4h13v95) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-5wyefd2bvxbt9w3nza3367af6evcjf4t4hf906r6zh5fc2dtwp-12i6mwlfqhhr2ighsbtxnan
sg 192.168.0.28:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

Creating Node2 as a Worker node:

```
[node2] (local) root@192.168.0.27 ~
$ docker swarm join --token SWMTKN-1-5wyefd2bvxbt9w3nza3367af6evcjf4t4hf906r6zh5fc2dtwp-12i6mwlfqhhr2ighsbtxnansg
 192.168.0.28:2377
This node joined a swarm as a worker.
```

Deploying the Docker stack:

```
[node1] (local) root@192.168.0.28 ~/example-voting-app
$ docker stack deploy --compose-file docker-stack.yml test
Creating network test_backend
Creating network test_frontend
Creating service test_db
Creating service test_vote
Creating service test_result
Creating service test_worker
Creating service test_redis
```

Here docker-stack.yml = Docker stack file.
test = Application name

Services created = 5
$docker stack ls

```
$ docker stack ls
NAME       SERVICES    ORCHESTRATOR
test       5           Swarm
```

$docker stack services test

```
[node1] (local) root@192.168.0.28 ~/example-voting-app
$ docker stack services test
ID             NAME          MODE         REPLICAS   IMAGE                                          PORTS
61yw9pkjmrzv   test_db       replicated   1/1        postgres:15-alpine
hn71uv2oeg1t   test_redis    replicated   1/1        redis:alpine
xl908w7hb8f8   test_result   replicated   1/1        dockersamples/examplevotingapp_result:latest   *:5001->80/tcp
1jj5ujv5bhu8   test_vote     replicated   2/2        dockersamples/examplevotingapp_vote:latest     *:5000->80/tcp
t8hxuwlswol9   test_worker   replicated   2/2        dockersamples/examplevotingapp_worker:latest
```

Containers Created = 7

```
$ docker ps
CONTAINER ID   IMAGE                                          COMMAND                  CREATED         STATUS
     PORTS      NAMES
592d246b46f4   dockersamples/examplevotingapp_result:latest   "/usr/bin/tini -- no…"   2 minutes ago   Up 2 minut
es   80/tcp     test_result.1.xbnzik37149y7m1advdxd093z
7992dc9225ee   postgres:15-alpine                             "docker-entrypoint.s…"   2 minutes ago   Up 2 minut
es   5432/tcp   test_db.1.etrjcd06ym300hfsl5tkdffon
e71a272a1b78   dockersamples/examplevotingapp_vote:latest     "gunicorn app:app -b…"   2 minutes ago   Up 2 minut
es   80/tcp     test_vote.1.bd84899tovcqqhc7ciusknb0r
717c312434a5   dockersamples/examplevotingapp_vote:latest     "gunicorn app:app -b…"   2 minutes ago   Up 2 minut
es   80/tcp     test_vote.2.rzeyld84ermd1xxv5dwnkpkm8
9b71161fc7ad   dockersamples/examplevotingapp_worker:latest   "dotnet Worker.dll"      2 minutes ago   Up 2 minut
es             test_worker.2.i50r6uyzzx842kxxm58cnwxpm
7e6e6dbf2a90   dockersamples/examplevotingapp_worker:latest   "dotnet Worker.dll"      2 minutes ago   Up 2 minut
es             test_worker.1.rbxs4isynlsugduu8bi9jku3g
c2bc16596207   redis:alpine                                   "docker-entrypoint.s…"   2 minutes ago   Up 2 minut
es   6379/tcp   test_redis.1.5k1pdn3rdmr921wap8u9t2ekd
```

$docker stack ps test

Containers distributed on Node1 and Node2

```
$ docker stack ps test
ID             NAME                IMAGE                                          NODE      DESIRED STATE   CURRE
NT STATE           ERROR                       PORTS
khta72sal4x4   test_db.1           postgres:15-alpine                             node1     Running         Runni
ng 23 minutes ago
7ajvh5ix2yxo   test_redis.1        redis:alpine                                   node2     Running         Runni
ng 23 minutes ago
mw7r7ksqzhsn   test_result.1       dockersamples/examplevotingapp_result:latest   node1     Running         Runni
ng 23 minutes ago
kennj7wkekde   test_vote.1         dockersamples/examplevotingapp_vote:latest     node2     Running         Runni
ng 23 minutes ago
zms9jqqvt0qu   test_vote.2         dockersamples/examplevotingapp_vote:latest     node1     Running         Runni
ng 23 minutes ago
yykuekqg6adi   test_worker.1       dockersamples/examplevotingapp_worker:latest   node2     Running         Runni
ng 23 minutes ago
mgs5duw1jiw5    \_ test_worker.1   dockersamples/examplevotingapp_worker:latest   node2     Shutdown        Faile
d 23 minutes ago    "task: non-zero exit (1)"
ap0d7f6gz0kt   test_worker.2       dockersamples/examplevotingapp_worker:latest   node1     Running         Running 23 minutes ago

1grp83vdl23u    \_ test_worker.2   dockersamples/examplevotingapp_worker:latest   node1     Shutdown        Failed 23 minutes ago    "task: non-ze
ro exit (1)"
```

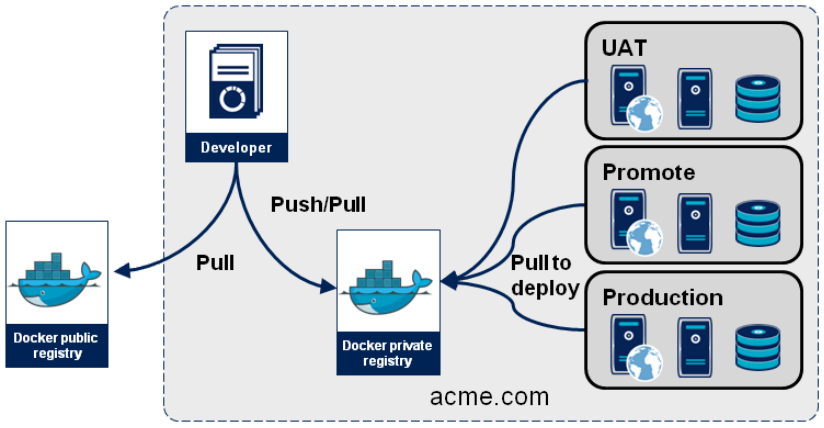
This is what Docker stack is.

DOCKER REGISTRY

The Registry is a stateless, highly scalable server side application that stores and lets us distribute Docker images.
The Docker Registry is open-source.

We should use the Registry if we want to:
- tightly control where our images are being stored
- fully own our images distribution pipeline
- integrate image storage and distribution tightly into our in-house development workflow

The Registry is compatible with Docker engine version 1.6.0 or higher.



PODMAN

INTRODUCTION

Podman (Pod Manager) is a fully featured container engine that is a simple **daemonless** tool. Podman provides a Docker-CLI comparable command line that eases the transition from other container engines and allows the management of pods, containers and images.
Podman manages the entire container ecosystem which includes pods, containers, container images, and container volumes using the **libpod library.**

Podman is an alias of Docker

FEATURES OF PODMAN

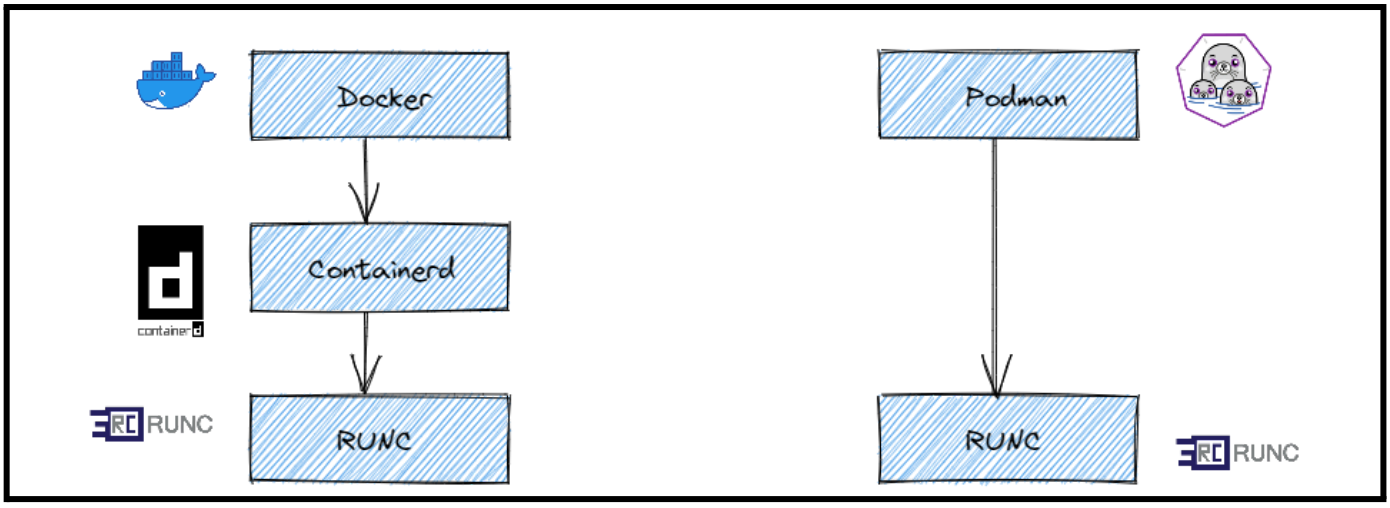
Daemonless
Podman stands out from other container engines because it is daemonless.
Daemons are processes that run in the background of our system to do the heavy lifting of running containers without a user interface.
Podman does not rely on a daemon to develop, manage or run containers. Since Podman does not have a daemon to manage containers, Podman uses another service manager to manage all of the services and support running containers in the background called Systemd.

Rootless / Supports Non-root
Rootless containers are containers that can be created, run, and managed by users without admin rights. Rootless containers have several advantages: They add a new security layer; even if the container engine, runtime, or orchestrator is compromised, the attacker won't gain root privileges on the host.
With Podman, users can run any container as non-root.

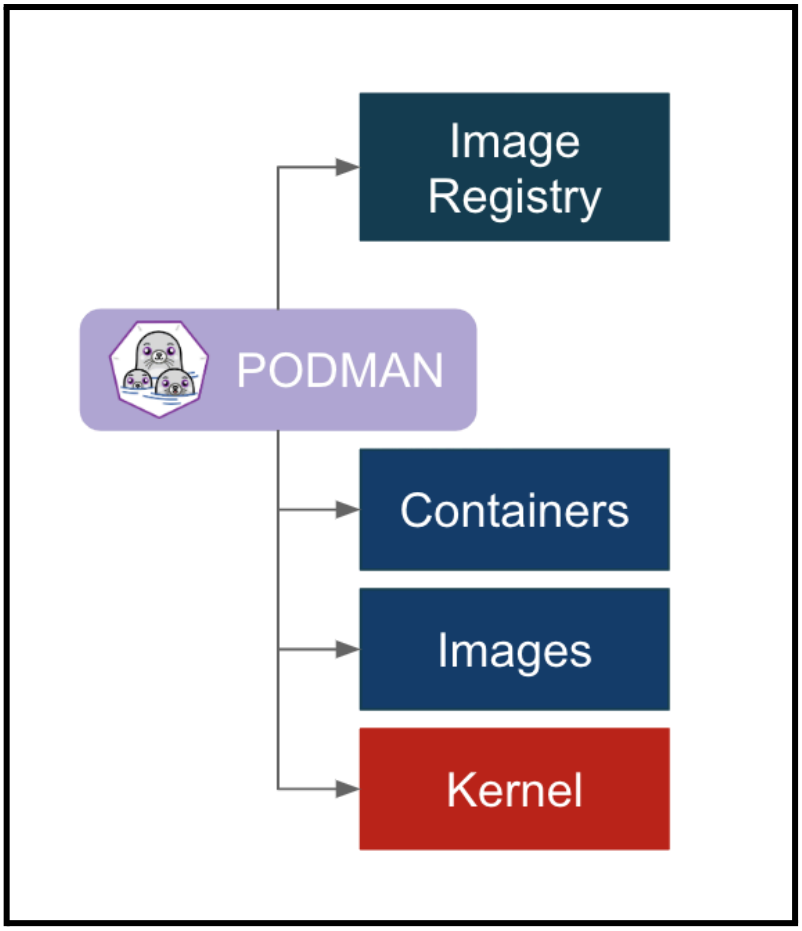
OpenSource
Podman (the POD manager) is an open source tool for developing, managing, and running containers on Linux systems.

Supports OCI Standard
OCI = Open Container Initiative
The Open Container Initiative is an open governance structure for the express purpose of creating open industry standards around container formats and runtimes.

Uses runC to deal with Containers
Podman and Buildah use runC, the OCI runtime, by default to launch containers.



PODMAN ARCHITECTURE



Podman uses Python language.
Docker is built on Go language.

HOW DOES PODMAN WORK?

Users can invoke Podman from the command line to pull containers from a repository and run them.
By integrating Systemd and Podman, we can generate control units for our containers and run them with Systemd automatically enabled.

References:
https://docs.podman.io/
https://www.redhat.com/
https://opencontainers.org/

KUBERNETES

K8s

CONTAINERS
Containers are a light-weight trimmed down version of the **operating system.**

ORCHESTRATION
Orchestration refers to the process of **managing and coordinating** multiple containers running in a distributed environment.
Container orchestration involves automating the deployment, scaling, and management of containers, as well as ensuring that they are running correctly and communicating with each other effectively.

Orchestration tools like **Docker Swarm and Kubernetes** provide a way to manage and automate the deployment and management of containers across a cluster of machines.

3 ADVANTAGES OF ORCHESTRATION

1. Centralised Management
2. High Availability
3. Load Balancing

of an application which is a **microservice** application, that is containerized.

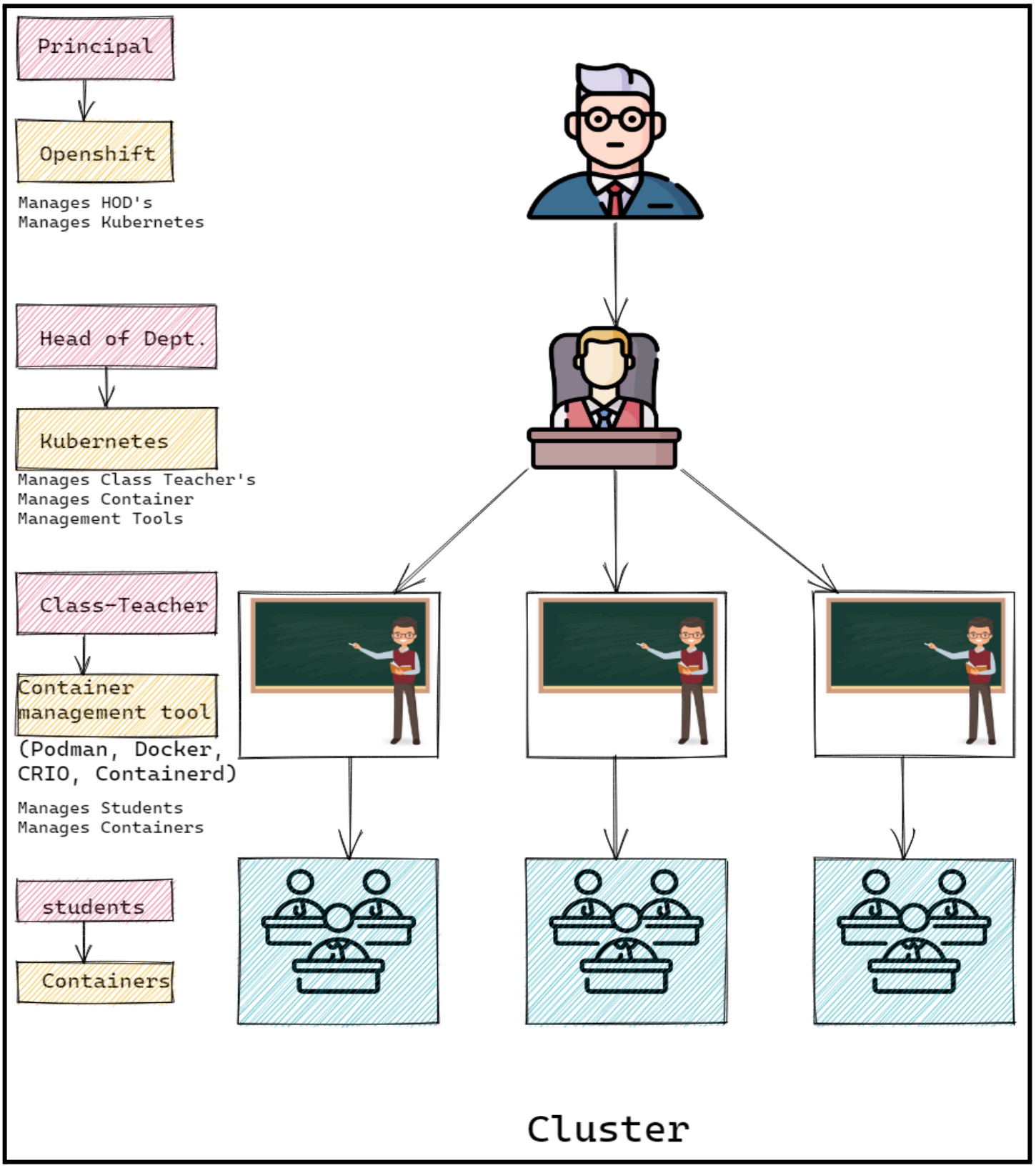
KUBERNETES
Kubernetes, also known as **K8s**, is an open-source, portable, extensible platform for managing containerized workloads and service systems, automating deployment, scaling, and management of containerized applications.

In Kubernetes, there is **NO Container Ecosystem**. We have a **Pod Ecosystem** here.
Pods in a cluster in Kubernetes can talk to each other because they have a **shared network namespace.**

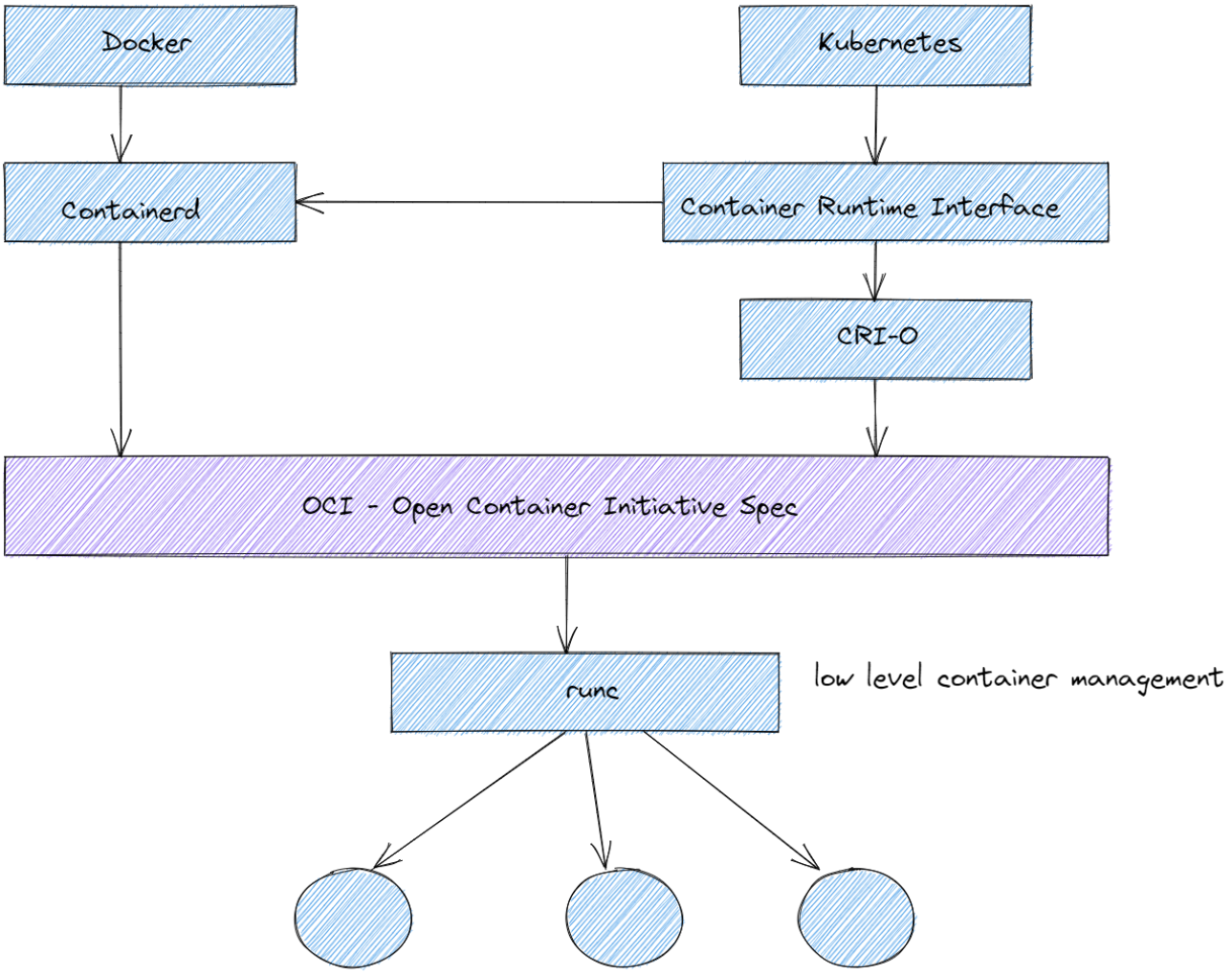
The name Kubernetes originates from **Greek**, meaning **helmsman or pilot.**
K8s as an abbreviation results from counting the eight letters between the "K" and the "s".

Kubernetes builds upon years of experience at **Google** running production workloads at scale using a system called **Borg**, combined with best-of-breed ideas and practices from the community. **Google released Kubernetes in 2014**, donated it as a seed technology for the founding of the **Cloud Native Computing Foundation in 2015**, and continues actively using and developing the project.

Best Example by **Ashutosh S. Bhakare Sir.**



COMPARING RUNTIMES



A **container runtime,** also known as **container engine**, is a software component that can run containers on a host operating system. In a containerized architecture, container runtimes are responsible for loading container images from a repository, monitoring local system resources, isolating system resources for use of a container, and managing container lifecycle.
Common examples of container runtimes are runC, Containerd, Docker, CRIO and rkt(rocket).
The **Open Container Interface (OCI)** is a Linux Foundation project started by **Docker**, which aims to provide open standards for Linux containers.

Note:
Docker was conquered by Mirantis
Red Hat was conquered by IBM; for acquiring Openshift
Google Anthos
Suse Rancher
(anthos and rancher can handle multi-cluster environment)

3 Types of Container Runtimes

a. Low - Level Container Runtime:
Low-level runtimes are responsible for creating and running containers. Once the containerized process runs, the container runtime is not required to perform other tasks. This is because low-level runtimes abstract the Linux primitives and are not designed to perform additional tasks.
Most popular Low-Level Container Runtimes are:

=> runC :
Created by Docker and the OCI.
runC is written in Go.

=> **crun:**
An OCI implementation led by Redhat.
crun is written in C.
Designed to be lightweight and performant, and was among the first runtimes to support cgroups v2.

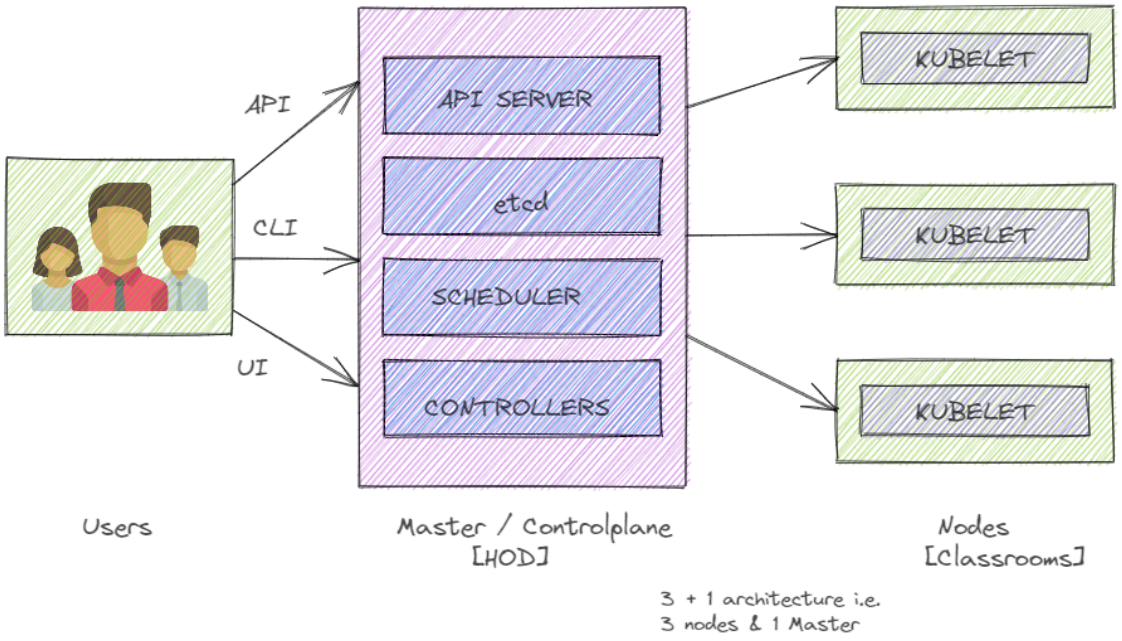
=> **Containerd :**
An open-source daemon supported by Linux and Windows, which facilitates the management of container life cycles through API requests.
The Containerd API adds a layer of abstraction and enhances container portability.

b. High - Level Container Runtime:
=> **Docker (Containerd) :**
It is the container runtime, providing image specifications, a command-line interface (CLI) and a container image-building service.

=> **CRI-O :**
An open-source implementation of Kubernetes' container runtime interface (CRI), offering a lightweight alternative to Docker. It allows running pods using OCI-compatible runtimes.

NOTE:
Now, Kubernetes works on CRIO or Containerd runtimes rather than docker, because docker doesn't comply with CRI.
So, K8s kicked out Docker.

KUBERNETES ARCHITECTURE [BIRD VIEW]



There are 3 ways to talk with Kubernetes:
a. API Calls
b. Command Line Interface
c. User Interface

COMPONENTS OF KUBERNETES [Bird View]

Kubelet

Kubelet is a **service**.
Kubelet is an agent / service which talks to the **master** through **API calls**.
It runs on all nodes, to see if the nodes are running or not.
It sends **heartbeats** ❤️ to the master when everything is good.
Kubelet also runs on the **master node.**
[says "**mai abhi jinda hu**" or **"kuch toh gadbad hai boss**"]
Theoretically, **1 node** can contain **110 Pods**, and pods in return consist of **Containers.**

Apiserver

It is the software that does the **authentication & authorization** of all the requests that strikes to the master.
It either accepts or rejects the requests.
It runs as a **Pod** in the Kubernetes cluster.

Example: Apiserver is the **Personal Assistant (PA)** of the HOD i.e. Kubernetes.
Example: krishna bhaiya of HOD mam.

Etcd

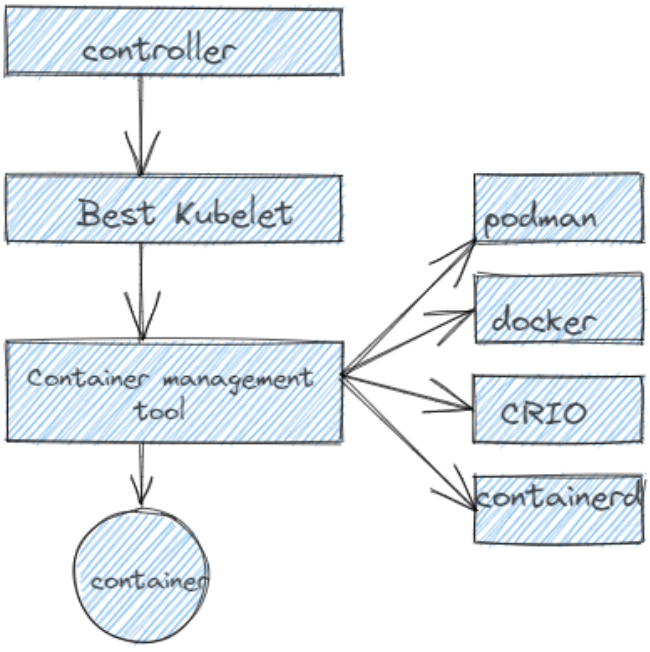
It is a component, which stores all the **metadata** about the cluster, the pods, nodes, etc.
The data stored is in the form of a **Key-value** database.
It runs as a **Pod** in the Kubernetes cluster.
Example: Etcd is the **Admission register** of college.

Scheduler

Scheduler holds the **election** of all nodes based on the criteria of **CPU and Memory**.
It elects the **best Node / Kubelet** for creating the new containers.
Scheduler **never runs the pods.** It only **selects** the node on which the pod will run.
It runs as a **Pod** in the Kubernetes cluster.

Controller

It is the component which talks to the **Best Kubelet / node** through **apiserver**, thereby **container management tools** like docker, crio, containerd, etc; which then creates the **containers**.
It runs as a **Pod** in the Kubernetes cluster.

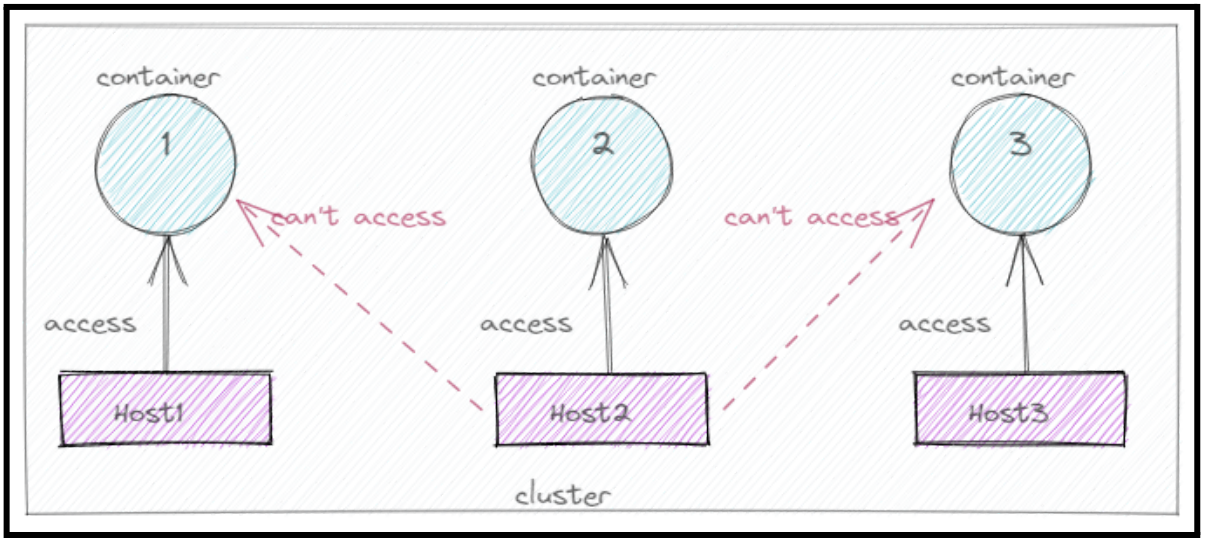


Note:
Pods by default are **Non-Persistent**.
When in a Kubelet cluster, any **master** crashes down, there is no high-availability by default.
We need to manually deploy / configure the **Multi-master cluster.**

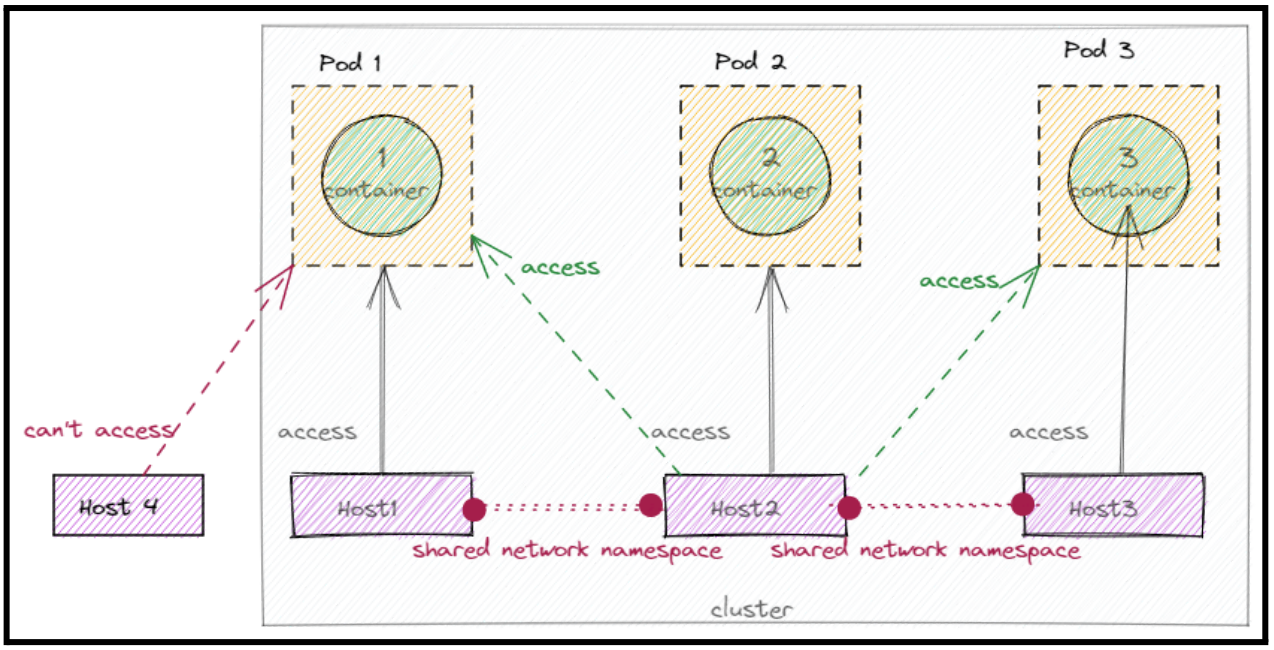
POD

Pod is a **wrapper** on top of a container.
A Pod (as in a pod of whales or pea pod) is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers.
A Pod is similar to a set of containers with shared namespaces and shared file system volumes.

Why Pod?



Here, we have **3 hosts**, and each host has a **container** inside them.
Host1 can access only the **container1**, and **can't** access **container2 or container3** as there is **no connection** between them. Same for Host 2 and Host3.
Therefore, to solve this issue, we use **PODS.**
Pods in a cluster can talk to each other, as they have the shared network namespace.



Here, **Host 4** is not part of the kubernetes cluster. Therefore, it is not connected with the network namespace. And so, **Host4 can't access the Pods.**

Kubernetes Basic Commands

We use the command "**kubectl**".
kubectl controls the Kubernetes cluster manager.

$kubectl get nodes
Shows the list of nodes / kubelets in a kubernetes cluster

```
controlplane $ kubectl get nodes
NAME           STATUS   ROLES           AGE   VERSION
controlplane   Ready    control-plane   30d   v1.26.1
node01         Ready    <none>          30d   v1.26.1
```

Here, we have 2 nodes in the cluster.
Controlplane: Master node
Node01: Worker node

$kubectl get nodes -o wide
Shows a brief descriptive information about the node in the cluster.

```
controlplane $ kubectl get nodes -o wide
NAME           STATUS   ROLES           AGE   VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION
  CONTAINER-RUNTIME
controlplane   Ready    control-plane   30d   v1.26.1   172.30.1.2    <none>        Ubuntu 20.04.5 LTS   5.4.0-131-generic
  containerd://1.6.12
node01         Ready    <none>          30d   v1.26.1   172.30.2.2    <none>        Ubuntu 20.04.5 LTS   5.4.0-131-generic
  containerd://1.6.12
```

Here, it is clear that **Kubernetes no longer uses docker as its runtime. Rather it uses Containerd.**

$kubectl get pods

Shows the list of created pods in the cluster. Here we haven't created any.

$kubectl get pods -n kube-system

Shows the list of default pods in the cluster. This contains all the components of the Kubernetes cluster. We have learned 4 components till today, they run as a pod in the kubernetes cluster.

-n: namespace. It is used for logical isolation.

```
controlplane $ kubectl get pods
No resources found in default namespace.
controlplane $ kubectl get pods -n kube-system -o wide
NAME                                       READY   STATUS    RESTARTS       AGE   IP              NODE           NOMINATED NO
DE   READINESS GATES
calico-kube-controllers-5f94594857-smxtv   1/1     Running   3 (12m ago)    30d   192.168.0.3     controlplane   <none>
     <none>
canal-ltd77                                2/2     Running   0              11m   172.30.2.2      node01         <none>
     <none>
canal-rzs9p                                2/2     Running   0              11m   172.30.1.2      controlplane   <none>
     <none>
coredns-68dc769db8-6cwk7                   1/1     Running   0              30d   192.168.0.7     controlplane   <none>
     <none>
coredns-68dc769db8-9h8gn                   1/1     Running   0              30d   192.168.1.2     node01         <none>
     <none>
etcd-controlplane                          1/1     Running   0              30d   172.30.1.2      controlplane   <none>
     <none>
kube-apiserver-controlplane                1/1     Running   2              30d   172.30.1.2      controlplane   <none>
     <none>
kube-controller-manager-controlplane       1/1     Running   3 (9m1s ago)   30d   172.30.1.2      controlplane   <none>
     <none>
kube-proxy-ldwqn                           1/1     Running   0              30d   172.30.2.2      node01         <none>
     <none>
kube-proxy-pftdf                           1/1     Running   0              30d   172.30.1.2      controlplane   <none>
     <none>
kube-scheduler-controlplane                1/1     Running   3 (12m ago)    30d   172.30.1.2      controlplane   <none>
     <none>
```

$kubectl cluster-info

Shows the details of the cluster.

```
controlplane $ kubectl cluster-info
Kubernetes control plane is running at https://172.30.1.2:6443
CoreDNS is running at https://172.30.1.2:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

.kube/config

Configuration file of Kubernetes cluster

```
controlplane $ cat .kube/config
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUMvakNDQWVhZ0F3SUJBZ0lCQURBTkJna3Foa2lHOXcwQkFRc0ZBREFWTVJNd0VRWURW
UVFERXdwcmRXSmwKY201bGRHVnpNQjRYRFRJek1EUXhNVEV6TkRZMU5Gb1hEVE16TURRd09ERXpORFkxTkZvd0ZURVRNQkVHQTFVRQpBeE1LYTNWaVpYSnVaWFJsY3pDQ0FTSXdEUVlk
S29aSWh2Y05BUUVCQlFBRGdnRVBBRENDQVFvQ2dnRUJBSi9XCm40OEh4d2gyWHZCdEVxZUxMSXBDWWFQbCs0S2VqYTFUUm41QkhNR2ZRTHRhdHorZXNyZjVaZ0JnMUNTaU00VmMKQkJt
SXMybkE1M044SG8rRUVMWnhTaDBLNk82SmtYTDFzK0FZWGN6RDZZMkRTVXVrU0RvSVdOV1hVNjh2cnpYbwpBS3dIVGxORWttaVZvbmhmeUN1MHBhN0M4ZmlhT2VRaXMrUzA0ekZHNFFF
NXlPY1JLalVOZCs4c01NalRrL0N6Cm81QUs0aHVoblZ0aVdYMzFJdENtallrT09tV1hQa0tnelloK2gxdXJqMm0vNFVrcEU4cXhoYkJBckwvOFFBTWkKS2RBUTkxdjZBQzBkd0NPeUdh
NGxCM2VjNDRqU3BDU003eU5heWw0SlpodnIzUDRtbGR4NXhwd1lTS0YzcTR5MwpUc2pFMmw2YXVBcWtlZVJEbnc4Q0F3RUFBYU5aTUZjd0RnWURWUjBQQVFIL0JBUURBZ0trTUE4R0Ex
VWRFd0VCCi93UUZNQU1CQWY4d0hRWURWUjBPQkJZRUZGenltY0VzSlZSc09kRlczSUwxaXd2eExaZHZNQlVHQTFVZEVRUU8KTUF5Q0NtdDFZbVZ5Ym1WMFpYTXdEUVlKS29aSWh2Y05B
UUVMQlFBRGdnRUJBSm9QZGk1anhCZzdOWE1heU1zUQozR2dKK1lwK2duQnR4VGFtVDZ0bENJendEZjRyL1Z3SWZCSkJaejBzUDRrQTFqUHp4NGhscEdwL01aelNUNVIzClF0clpod1JN
UWtQdGVyWmlUUTdTUmJNT014djZIR3dsdlptSWhVUGZYbnBNWXJvWTFCbGJkQ0xYL1F2eHE5UlMKS1hJVWpjOW96T3hFOHNqZkFQbTczOEx5OU1TQUpFbEdyQytQU1JMTEVxUkg0MUo0
eHNnOWFGZGZuL3d5bmh4SQpLZHhNdDlsRjVyRjRURW03Qlk3VTZIT3VJZWJ2b1kzc0l0N2NId05BeCsyeVdrc2tWWm9VdUJ4dWNMc2FFdWRGCjJDRSt2dG1GdXBqQ2dLVDVjaC9kYm1r
OG9PYnNoVUcvUVVSdktyQ1Ftc2wyNFBuUHRlVmlxemp4MHFkdlJqNFMKdlJNPQotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
    server: https://172.30.1.2:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURJVENDQWdtZ0F3SUJBZ0lJR0FtVDZyZ2Z6aWd3RFFZSktvWklodmNOQVFFTEJRQXdGVEV
```

The worker node “node01” can’t view the list of pods or nodes. It can only be viewed by the **master**.

```
controlplane $ ssh node01
Last login: Sun Nov 13 17:27:09 2022 from 10.48.0.33
node01 $ ls
filesystem
node01 $ kubectl get nodes
E0512 04:03:28.177577   37309 memcache.go:238] couldn't get current server API group list: Get "http://localhost:8080/api?timeout=32s": dial
 tcp 127.0.0.1:8080: connect: connection refused
E0512 04:03:28.178077   37309 memcache.go:238] couldn't get current server API group list: Get "http://localhost:8080/api?timeout=32s": dial
 tcp 127.0.0.1:8080: connect: connection refused
E0512 04:03:28.179383   37309 memcache.go:238] couldn't get current server API group list: Get "http://localhost:8080/api?timeout=32s": dial
 tcp 127.0.0.1:8080: connect: connection refused
E0512 04:03:28.180721   37309 memcache.go:238] couldn't get current server API group list: Get "http://localhost:8080/api?timeout=32s": dial
 tcp 127.0.0.1:8080: connect: connection refused
E0512 04:03:28.182055   37309 memcache.go:238] couldn't get current server API group list: Get "http://localhost:8080/api?timeout=32s": dial
 tcp 127.0.0.1:8080: connect: connection refused
The connection to the server localhost:8080 was refused - did you specify the right host or port?
node01 $ kubectl get pods
E0512 04:03:38.365940   37372 memcache.go:238] couldn't get current server API group list: Get "http://localhost:8080/api?timeout=32s": dial
 tcp 127.0.0.1:8080: connect: connection refused
E0512 04:03:38.366268   37372 memcache.go:238] couldn't get current server API group list: Get "http://localhost:8080/api?timeout=32s": dial
 tcp 127.0.0.1:8080: connect: connection refused
E0512 04:03:38.367560   37372 memcache.go:238] couldn't get current server API group list: Get "http://localhost:8080/api?timeout=32s": dial
 tcp 127.0.0.1:8080: connect: connection refused
E0512 04:03:38.368837   37372 memcache.go:238] couldn't get current server API group list: Get "http://localhost:8080/api?timeout=32s": dial
 tcp 127.0.0.1:8080: connect: connection refused
E0512 04:03:38.370209   37372 memcache.go:238] couldn't get current server API group list: Get "http://localhost:8080/api?timeout=32s": dial
 tcp 127.0.0.1:8080: connect: connection refused
The connection to the server localhost:8080 was refused - did you specify the right host or port?
```

Creating a Kubernetes Pod using YAML file

Every YAML file has **4** sections:
a. **apiVersion** : version
b. **kind** : object
c. **metadata** : labels
d. **spec** : specification

Labels in yaml files are used for tagging, grouping, sorting or for mapping purposes.

Labels can be **meaningful or meaningless**.
They are in the form of **key-value** pairs.

$vi pod.yml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod1
  labels:
    class: first_year
    division: A
spec:
  containers:
  - name: container1
    image: docker.io/httpd
    ports:
    - containerPort: 80
```

[Note: Everything needs to be written in small case]
This is the **yaml** file which will create the POD.
Here, name of **pod** = **Pod1**
Labels have 2 pairs: **class : First year** and **division : A**.
The container name = **Container**1. This container will be created inside the pod.
Image = **httpd** from docker hub.

Creating pod using this pod.yml file: Absolute way
$kubectl create -f <filename.yml>

```
controlplane $ #vi pod.yml
controlplane $ kubectl create -f pod.yml
pod/pod1 created
```

Listing the pod:

```
controlplane $ kubectl get pods
NAME   READY   STATUS    RESTARTS   AGE
pod1   1/1     Running   0          75s
```

```
controlplane $ kubectl get pods -o wide
NAME   READY   STATUS    RESTARTS   AGE    IP            NODE     NOMINATED NODE   READINESS GATES
pod1   1/1     Running   0          100s   192.168.1.3   node01   <none>           <none>
```

$kubectl get pods –show-labels
This will list the labels

```
controlplane $ kubectl get pods --show-labels
NAME   READY   STATUS    RESTARTS   AGE     LABELS
pod1   1/1     Running   0          3m39s   class=first_year,division=A
```

We can create multiple pods using the same pod.yml file, but with **different pod names.**

```
controlplane $ vi pod.yml
controlplane $ kubectl create -f pod.yml
pod/pod2 created
controlplane $ kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
pod1    1/1     Running   0          5m17s
pod2    1/1     Running   0          7s
```

If we delete the pod.yml file, the pods remain constant

```
controlplane $ ls
filesystem  pod.yml
controlplane $ rm pod.yml
controlplane $ ls
filesystem
controlplane $ kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
pod1    1/1     Running   0          6m31s
pod2    1/1     Running   0          81s
```

Changing the labels of pod.yml file

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod3
  labels:
    class: second_year
    division: A
spec:
  containers:
  - name: mycontainer1
    image: docker.io/httpd
    ports:
    - containerPort: 80
```

Creating a pod using the pod.yml file

```
controlplane $ kubectl create -f pod.yml
pod/pod3 created
controlplane $ kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
pod1    1/1     Running   0          9m44s
pod2    1/1     Running   0          4m34s
pod3    1/1     Running   0          5s
```

Using the selector parameter "**-l**" for segregating / mapping pods based on some selection criteria.

```
controlplane $ kubectl get pods -l class=first_year
NAME    READY   STATUS    RESTARTS   AGE
pod1    1/1     Running   0          10m
pod2    1/1     Running   0          5m20s
controlplane $ kubectl get pods -l class=second_year
NAME    READY   STATUS    RESTARTS   AGE
pod3    1/1     Running   0          68s
controlplane $ kubectl get pods -l division=A
NAME    READY   STATUS    RESTARTS   AGE
pod1    1/1     Running   0          10m
pod2    1/1     Running   0          5m46s
pod3    1/1     Running   0          77s
```

$kubectl describe pod <pod_name>

Describes the pod (same as inspect)

```
controlplane $ kubectl describe pod pod1
Name:             pod1
Namespace:        default
Priority:         0
Service Account:  default
Node:             node01/172.30.2.2
Start Time:       Fri, 12 May 2023 04:13:30 +0000
Labels:           class=first_year
                  division=A
Annotations:      cni.projectcalico.org/containerID: 1e4935347728813355ce9c1894ca1a8af2b7bf0c64e9fcf331cf28882c7cde1a
                  cni.projectcalico.org/podIP: 192.168.1.3/32
                  cni.projectcalico.org/podIPs: 192.168.1.3/32
Status:           Running
IP:               192.168.1.3
IPs:
  IP:  192.168.1.3
Containers:
  container1:
    Container ID:   containerd://ab2deaef319c91c1afc9c5eadec61e192d7c0c3cb647b70111fca7506e839b9a
    Image:          docker.io/httpd
    Image ID:       docker.io/library/httpd@sha256:90254ccc7352e1a5c8d1e4cdab2a032cefac9fd5d4d632ca003a2943c9a9b0a3
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Fri, 12 May 2023 04:13:38 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-8flnh (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
```

Events that take place while container Creation

```
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  11m   default-scheduler  Successfully assigned default/pod1 to node01
  Normal  Pulling    11m   kubelet            Pulling image "docker.io/httpd"
  Normal  Pulled     11m   kubelet            Successfully pulled image "docker.io/httpd" in 7.983881784s (7.9838874s including waiting)
  Normal  Created    11m   kubelet            Created container container1
  Normal  Started    11m   kubelet            Started container container1
```

Scheduler : elects the best kubelet / node.

Pulling of images from public/private registry.

Image **pulled**.

Creating the container using the pulled image.
Starting the container.

$kubectl exec -it <podname> <shell>
Getting inside the container of Pod "Pod1"
-i : interactive
-t : tty
Changing the default index.html page's information.

```
controlplane $ kubectl exec -it pod1 bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] --
 [COMMAND] instead.
root@pod1:/usr/local/apache2# ls
bin  build  cgi-bin  conf  error  htdocs  icons  include  logs  modules
root@pod1:/usr/local/apache2# cd htdocs/
root@pod1:/usr/local/apache2/htdocs# ls
index.html
root@pod1:/usr/local/apache2/htdocs# cat index.html
<html><body><h1>It works!</h1></body></html>
root@pod1:/usr/local/apache2/htdocs# cat > index.html
<html><body><h1>Hello There!!</h1></body></html>


root@pod1:/usr/local/apache2/htdocs# cat index.html
<html><body><h1>Hello There!!</h1></body></html>


root@pod1:/usr/local/apache2/htdocs#
```

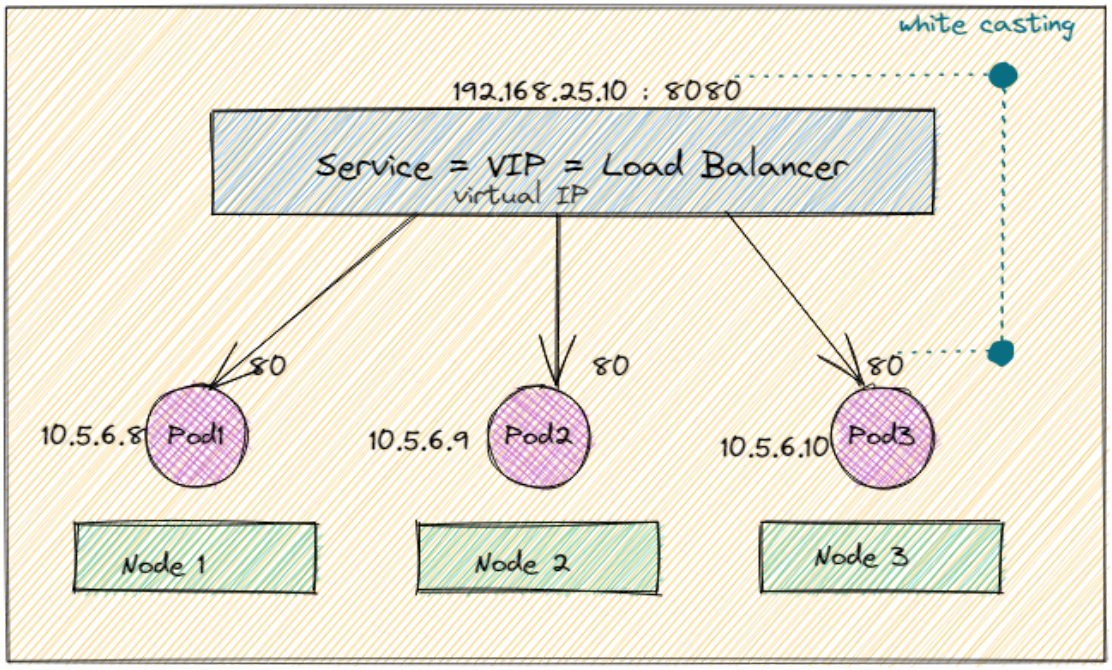
This pod has just 1 container, therefore we have not mentioned the container name in which we need to enter.
But if our pod has more than 1 container inside it, then we need to specify the container name using the argument "**-c**" in which we want to enter.

```
controlplane $ kubectl get pods -o wide
NAME   READY   STATUS    RESTARTS   AGE     IP            NODE     NOMINATED NODE   READINESS GATES
pod1   1/1     Running   0          16m     192.168.1.3   node01   <none>           <none>
pod2   1/1     Running   0          10m     192.168.1.4   node01   <none>           <none>
pod3   1/1     Running   0          6m30s   192.168.1.5   node01   <none>           <none>
controlplane $ curl 192.168.1.3
<html><body><h1>Hello There!!</h1></body></html>
```

Problem Statement
We have 5 pod replicas, which contain the same/ different applications.
If one pod goes down, we need to change the reachable IP **manually**.
If the pods go down more frequently, then it will become a tedious task to manually change the IP every time.
Pods in a cluster have **dynamic IP addresses. [changing ip addresses]**
We need a **SPOC IP (single point of contact)** to access the application, rather than having multiple IP's.

SERVICE

Here comes the Hero of our cluster**: SERVICE.**
Service is an object / logic which **groups the pods together.**
It is a method for exposing a network application that is running as one or more Pods in our cluster.
Service = Central Point.
Example: **Helpline number** of any Institute / Organization



If one pod goes down, then it will **automatically** move on the next pod's IP.
If all pods are in good condition, then while **load balancing** the **RR method i.e. Round Robin algorithm** will be followed.
In a round robin arrangement, for instance, if we have five pods, the load balancer will send the first request to pod 1, the second request to pod 2, and so on in a **recursive cycle.**
Therefore, here rather than talking to the pods individually, we are talking **directly to the service.**

Services talk / map to the pods using "**SELECTORS**" that map to the labels of the pods.

There are **4 types of Services.**
1. CLUSTER IP (default)
Internal; Service can't be accessed outside the cluster.

Practical : Service: Cluster IP

Firstly, creating **3 pods** with **labels= state:maharashtra** & **2 pod**s with **labels=city:gujarat.**

$vi podfile.yml apiVersion: v1 kind: Pod metadata: name: pod1 labels: **state: maharashtra** spec: containers: - name: c1 image: docker.io/httpd ports: **- containerPort: 80**	**$vi podfile.yml** apiVersion: v1 kind: Pod metadata: name: pod4 labels: **state: gujarat** spec: containers: - name: c1 image: docker.io/httpd ports: **- containerPort: 80**

Creating Pods

```
controlplane $ vi podfile.yml
controlplane $ kubectl create -f podfile.yml
pod/pod1 created
controlplane $ vi podfile.yml
controlplane $ kubectl create -f podfile.yml
pod/pod2 created
controlplane $ vi podfile.yml
controlplane $ kubectl create -f podfile.yml
pod/pod3 created
controlplane $ kubectl get pods --show-labels -o wide
NAME   READY   STATUS    RESTARTS   AGE   IP             NODE     NOMINATED NODE   READINESS GATES   LABELS
pod1   1/1     Running   0          46s   192.168.1.3    node01   <none>           <none>            state=maharashtra
pod2   1/1     Running   0          29s   192.168.1.4    node01   <none>           <none>            state=maharashtra
pod3   1/1     Running   0          18s   192.168.1.5    node01   <none>           <none>            state=maharashtra
```

```
controlplane $ kubectl create -f podfile.yml
pod/pod4 created
controlplane $ vi podfile.yml
controlplane $ kubectl create -f podfile.yml
pod/pod5 created
controlplane $ kubectl get pods --show-labels -o wide
NAME   READY   STATUS    RESTARTS   AGE     IP             NODE     NOMINATED NODE   READINESS GATES   LABELS
pod1   1/1     Running   0          4m5s    192.168.1.3    node01   <none>           <none>            state=maharashtra
pod2   1/1     Running   0          3m48s   192.168.1.4    node01   <none>           <none>            state=maharashtra
pod3   1/1     Running   0          3m37s   192.168.1.5    node01   <none>           <none>            state=maharashtra
pod4   1/1     Running   0          20s     192.168.1.6    node01   <none>           <none>            state=gujarat
pod5   1/1     Running   0          4s      192.168.1.7    node01   <none>           <none>            state=gujarat
```

Creating 2 Services using YAML file

$vi svc.yml apiVersion: v1 kind: Service metadata: name: svc1	**$vi svc.yml** apiVersion: v1 kind: Service metadata: name: svc1

spec: selector: **state: maharashtra** ports: - protocol: TCP **port: 8080** **targetPort: 80**	spec: selector: **state: maharashtra** ports: - protocol: TCP **port: 8000** **targetPort: 80**

Service 1:

```
controlplane $ vi svc.yml
controlplane $ kubectl create -f svc.yml
service/svc1 created
controlplane $ kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP    31d
svc1         ClusterIP   10.102.183.11   <none>        8080/TCP   7s
controlplane $ kubectl get svc -o wide
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE   SELECTOR
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP    31d   <none>
svc1         ClusterIP   10.102.183.11   <none>        8080/TCP   14s   state=maharashtra
```

Here, the ClusterIP = VIP. This IP is the centrally reachable IP.

Let's get the detailed description of the svc1.

$kubectl describe svc svc1

```
controlplane $ kubectl describe svc svc1
Name:              svc1
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          state=maharashtra
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.102.183.11
IPs:               10.102.183.11
Port:              <unset>  8080/TCP
TargetPort:        80/TCP
Endpoints:         192.168.1.3:80,192.168.1.4:80,192.168.1.5:80
Session Affinity:  None
Events:            <none>
controlplane $ kubectl get pods -o wide
NAME   READY   STATUS    RESTARTS   AGE     IP            NODE     NOMINATED NODE   READINESS GATES
pod1   1/1     Running   0          13m     192.168.1.3   node01   <none>           <none>
pod2   1/1     Running   0          13m     192.168.1.4   node01   <none>           <none>
pod3   1/1     Running   0          13m     192.168.1.5   node01   <none>           <none>
pod4   1/1     Running   0          10m     192.168.1.6   node01   <none>           <none>
pod5   1/1     Running   0          9m44s   192.168.1.7   node01   <none>           <none>
```

The service **automatically groups the pods** having the label as the selector.

Port: is the Service Port number.

Target Port : is always the container's port number.

Port has an attribute "**unset**" => It is the name of the port.

Service 2:

```
controlplane $ vi svc.yml
controlplane $ kubectl create -f svc.yml
service/svc2 created
controlplane $ kubectl get svc -o wide
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE     SELECTOR
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP    31d     <none>
svc1         ClusterIP   10.102.183.11    <none>        8080/TCP   7m29s   state=maharashtra
svc2         ClusterIP   10.105.248.244   <none>        8000/TCP   10s     state=gujarat
```

$kubectl describe svc svc2

```
controlplane $ kubectl describe svc svc2
Name:              svc2
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          state=gujarat
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.105.248.244
IPs:               10.105.248.244
Port:              <unset>  8000/TCP
TargetPort:        80/TCP
Endpoints:         192.168.1.6:80,192.168.1.7:80
Session Affinity:  None
Events:            <none>
controlplane $ kubectl get pods -o wide --show-label
error: unknown flag: --show-label
See 'kubectl get --help' for usage.
controlplane $ kubectl get pods -o wide --show-labels
NAME   READY   STATUS    RESTARTS   AGE   IP            NODE     NOMINATED NODE   READINESS GATES   LABELS
pod1   1/1     Running   0          19m   192.168.1.3   node01   <none>           <none>            state=maharashtra
pod2   1/1     Running   0          18m   192.168.1.4   node01   <none>           <none>            state=maharashtra
pod3   1/1     Running   0          18m   192.168.1.5   node01   <none>           <none>            state=maharashtra
pod4   1/1     Running   0          15m   192.168.1.6   node01   <none>           <none>            state=gujarat
pod5   1/1     Running   0          15m   192.168.1.7   node01   <none>           <none>            state=gujarat
```

Creating another Pod “pod6” with label state:gujarat

```
controlplane $ vi podfile.yml
controlplane $ kubectl create -f podfile.yml
pod/pod6 created
```

Let’s describe the svc2, which has the selector as state:gujarat

```
controlplane $ kubectl get pods -o wide
NAME   READY   STATUS    RESTARTS   AGE    IP             NODE           NOMINATED NODE   READINESS GATES
pod1   1/1     Running   0          23m    192.168.1.3    node01         <none>           <none>
pod2   1/1     Running   0          22m    192.168.1.4    node01         <none>           <none>
pod3   1/1     Running   0          22m    192.168.1.5    node01         <none>           <none>
pod4   1/1     Running   0          19m    192.168.1.6    node01         <none>           <none>
pod5   1/1     Running   0          19m    192.168.1.7    node01         <none>           <none>
pod6   1/1     Running   0          2m3s   192.168.0.8    controlplane   <none>           <none>
controlplane $ kubectl describe svc svc2
Name:              svc2
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          state=gujarat
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.105.248.244
IPs:               10.105.248.244
Port:              <unset>  8000/TCP
TargetPort:        80/TCP
Endpoints:         192.168.0.8:80,192.168.1.6:80,192.168.1.7:80
Session Affinity:  None
Events:            <none>
```

The pod 6 is automatically added in the endpoints of the service.Thus, no manual work needed.

Deleting pod1:

```
controlplane $ kubectl delete pod pod1
pod "pod1" deleted
controlplane $ kubectl get pods
NAME   READY   STATUS    RESTARTS   AGE
pod2   1/1     Running   0          25m
pod3   1/1     Running   0          25m
pod4   1/1     Running   0          22m
pod5   1/1     Running   0          22m
pod6   1/1     Running   0          4m53s
```

Verifying the service endpoints of svc1:

```
controlplane $ kubectl describe svc svc1
Name:              svc1
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          state=maharashtra
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.102.183.11
IPs:               10.102.183.11
Port:              <unset>  8080/TCP
TargetPort:        80/TCP
Endpoints:         192.168.1.4:80,192.168.1.5:80
Session Affinity:  None
Events:            <none>
```

It has automatically removed the IP of Pod1 from the endpoints of service svc1.
Describing;

```
controlplane $ kubectl describe endpoints svc1
Name:         svc1
Namespace:    default
Labels:       <none>
Annotations:  <none>
Subsets:
  Addresses:          192.168.1.4,192.168.1.5
  NotReadyAddresses:  <none>
  Ports:
    Name     Port  Protocol
    ----     ----  --------
    <unset>  80    TCP

Events:  <none>
controlplane $ kubectl describe endpoints svc2
Name:         svc2
Namespace:    default
Labels:       <none>
Annotations:  endpoints.kubernetes.io/last-change-trigger-time: 2023-05-13T11:43:34Z
Subsets:
  Addresses:          192.168.0.8,192.168.1.6,192.168.1.7
  NotReadyAddresses:  <none>
  Ports:
    Name     Port  Protocol
    ----     ----  --------
    <unset>  80    TCP

Events:  <none>
```

Curling the Service IP's

```
controlplane $ kubectl get svc -o wide
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE   SELECTOR
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP    31d   <none>
svc1         ClusterIP   10.102.183.11    <none>        8080/TCP   17m   state=maharashtra
svc2         ClusterIP   10.105.248.244   <none>        8000/TCP   10m   state=gujarat
controlplane $ curl 10.102.183.11:8080
<html><body><h1>It works!</h1></body></html>
controlplane $ curl 10.105.248.244:8000
<html><body><h1>It works!</h1></body></html>
```

We can see the same output, as the pods contain the same httpd application.
Let's change some lines in the default index.html page.

```
controlplane $ kubectl exec -it pod2 bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] ins
tead.
root@pod2:/usr/local/apache2# cat > htdocs/index.html
Hello From Pod 2.
^C
root@pod2:/usr/local/apache2# exit
exit
command terminated with exit code 130
controlplane $ kubectl exec -it pod3 bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] ins
tead.
root@pod3:/usr/local/apache2# cat > htdocs/index.html
Hello From Pod3.^C
root@pod3:/usr/local/apache2# exit
exit
command terminated with exit code 130
```

```
controlplane $ kubectl exec -it pod4 bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] ins
tead.
root@pod4:/usr/local/apache2# cat > htdocs/index.html
Hello from Pod4
^C
root@pod4:/usr/local/apache2# exit
exit
command terminated with exit code 130
controlplane $ kubectl exec -it pod5 bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] ins
tead.
root@pod5:/usr/local/apache2# cat > htdocs/index.html
Hello from Pod5
^C
root@pod5:/usr/local/apache2# exit
exit
command terminated with exit code 130
controlplane $ kubectl exec -it pod6 bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] ins
tead.
root@pod6:/usr/local/apache2# cat htdocs/index.html
<html><body><h1>It works!</h1></body></html>
root@pod6:/usr/local/apache2# cat > htdocs/index.html
Hello From Pod6.^C
root@pod6:/usr/local/apache2# exit
```

Curling:

```
controlplane $ kubectl get svc -o wide
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE   SELECTOR
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP    31d   <none>
svc1         ClusterIP   10.102.183.11    <none>        8080/TCP   24m   state=maharashtra
svc2         ClusterIP   10.105.248.244   <none>        8000/TCP   17m   state=gujarat
controlplane $ curl 10.102.183.11:8080
Hello From Pod 2.
controlplane $ curl 10.102.183.11:8080
Hello From Pod 2.
controlplane $ curl 10.102.183.11:8080
Hello From Pod3.controlplane $ curl 10.102.183.11:8080
Hello From Pod 2.
controlplane $ curl 10.102.183.11:8080
Hello From Pod3.controlplane $
```

```
controlplane $ curl 10.105.248.244:8000
Hello from Pod5
controlplane $ curl 10.105.248.244:8000
Hello from Pod4
controlplane $ curl 10.105.248.244:8000
controlplane $ curl 10.105.248.244:8000
Hello from Pod5
controlplane $ curl 10.105.248.244:8000
Hello from Pod4
controlplane $ curl 10.105.248.244:8000
Hello from Pod4
controlplane $ curl 10.105.248.244:8000
Hello from Pod5
controlplane $
```

Deleting the Services

```
controlplane $ kubectl delete svc svc1 svc2
service "svc1" deleted
service "svc2" deleted
controlplane $ kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   31d
```

Example:
Creating a Service with Openshift Application.

$vi Pod.yml	**$vi svc.yml**

apiVersion: v1
kind: Pod
metadata:
  name: poda
  labels:
  course: cka
spec:
  containers:
  - name: c1
  image: docker.io/openshift/hello-openshift
  ports:
  - containerPort: **8080**
	apiVersion: v1
	kind: Service
	metadata:
	  name: my-svc
	spec:
	  selector:
	  course: kucl
	  ports:
	  - protocol: TCP
	  port: 8080
	  targetPort: 8080

Pod Creation:

```
controlplane $ vi pod.yml
controlplane $ kubectl create -f pod.yml
pod/poda created
controlplane $ kubectl get pods | grep poda
poda   1/1     Running   0          22s
controlplane $ kubectl get pods --show-labels -o wide | grep poda
poda   1/1     Running   0          37s   192.168.1.8   node01   <none>           <none>            course=cka
```

The label of the pod and the selector of service is different.

Service creation.

```
controlplane $ vi svc.yml
controlplane $ kubectl create -f svc.yml
service/my-svc created
```

Describing Service:

```
controlplane $ kubectl describe svc my-svc
Name:              my-svc
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          course=kucl
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.97.86.93
IPs:               10.97.86.93
Port:              <unset>  8080/TCP
TargetPort:        8080/TCP
Endpoints:         <none>
Session Affinity:  None
Events:            <none>
```

Here, we don't have any endpoint IP's as no pod has the **label = course: kucl.**

Therefore, let's create another **pod "podb"** with label as **course:kucl**

```
controlplane $ vi pod.yml
controlplane $ kubectl describe svc my-svc
Name:              my-svc
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          course=kucl
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.97.86.93
IPs:               10.97.86.93
Port:              <unset>  8080/TCP
TargetPort:        8080/TCP
Endpoints:         192.168.0.9:8080
Session Affinity:  None
Events:            <none>
```

Podb added to the service.

Editing the service

$kubectl edit svc <service_name>

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: "2023-05-13T12:11:45Z"
  name: my-svc
  namespace: default
  resourceVersion: "5929"
  uid: 1049c110-b173-4dd6-a3ff-cdde12ab2454
spec:
  clusterIP: 10.97.86.93
  clusterIPs:
  - 10.97.86.93
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 8080
  selector:
    course: kucl
  sessionAffinity: None
```

```
  selector:
    course: cka
  sessionAffinity: None
  type: ClusterIP
```

```
controlplane $ kubectl edit svc my-svc
service/my-svc edited
```

```
controlplane $ kubectl get pods --show-labels -o wide
NAME   READY   STATUS    RESTARTS   AGE     IP              NODE           NOMINATED NODE   READINESS GATES   LABELS
pod2   1/1     Running   0          54m     192.168.1.4     node01         <none>           <none>            state=maharasht
ra
pod3   1/1     Running   0          54m     192.168.1.5     node01         <none>           <none>            state=maharasht
ra
pod4   1/1     Running   0          51m     192.168.1.6     node01         <none>           <none>            state=gujarat
pod5   1/1     Running   0          51m     192.168.1.7     node01         <none>           <none>            state=gujarat
pod6   1/1     Running   0          33m     192.168.0.8     controlplane   <none>           <none>            state=gujarat
poda   1/1     Running   0          10m     192.168.1.8     node01         <none>           <none>            course=cka
podb   1/1     Running   0          3m11s   192.168.0.9     controlplane   <none>           <none>            course=kucl
controlplane $ kubectl describe svc my-svc
Name:              my-svc
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          course=cka
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.97.86.93
IPs:               10.97.86.93
Port:              <unset>  8080/TCP
TargetPort:        8080/TCP
Endpoints:         192.168.1.8:8080
Session Affinity:  None
Events:            <none>
```

Thus**, poda** is now **added** to the service, and **podb** is **removed** automatically.

Can a Service have 2 Target Ports? YES

Creating 2 pods with different ports.

apiVersion: v1
kind: Pod
metadata:
  name: pod1
  labels:
  course: cka
spec:
  containers:
  - name: c1
  image: docker.io/openshift/hello-openshift
  ports:
  - containerPort: 8080
	apiVersion: v1
	kind: Pod
	metadata:
	  name: pod2
	  labels:
	  course: cka
	spec:
	  containers:
	  - name: c1
	  image: docker.io/openshift/hello-openshift
	  ports:
	  - containerPort: 8888

```
controlplane $ vi pod1.yml
controlplane $ vi pod2.yml
controlplane $ kubectl create -f pod1.yml
pod/pod1 created
controlplane $ kubectl create -f pod2.yml
pod/pod2 created
controlplane $ kubectl get pods -o wide --show-labels
NAME   READY   STATUS    RESTARTS   AGE   IP              NODE     NOMINATED NODE   READINESS GATES   LABELS
pod1   1/1     Running   0          19s   192.168.1.3     node01   <none>           <none>            course=cka
pod2   1/1     Running   0          15s   192.168.1.4     node01   <none>           <none>            course=cka
```

Creating a Service

$vi svc.yml

```
apiVersion: v1
kind: Service
metadata:
  name: svc1
spec:
  selector:
    course: cka
  ports:
  - name: p1
    protocol: TCP
    port: 8080
    targetPort: 8080
  - name: p2
    protocol: TCP
    port: 8888
    targetPort: 8888
```

Here, we have mentioned 2 slots of ports for assigning 2 ports to the service.
Also, we have mentioned the name value for the ports, and therefore it took the place of **<unset>**

```
controlplane $ vi svc.yml
controlplane $ kubectl create -f svc.yml
service/svc1 created
controlplane $ kubectl describe svc svc1
Name:              svc1
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          course=cka
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.104.26.238
IPs:               10.104.26.238
Port:              p1  8080/TCP
TargetPort:        8080/TCP
Endpoints:         192.168.1.3:8080,192.168.1.4:8080
Port:              p2  8888/TCP
TargetPort:        8888/TCP
Endpoints:         192.168.1.3:8888,192.168.1.4:8888
Session Affinity:  None
Events:            <none>
```

Curling

```
controlplane $ kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)             AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP             31d
svc1         ClusterIP   10.104.26.238   <none>        8080/TCP,8888/TCP   2m37s
controlplane $ curl 10.104.26.238:8080
Hello OpenShift!
controlplane $ curl 10.104.26.238:8888
Hello OpenShift!
```

SERVICE HAVING TARGET PORTS OF DIFFERENT APPLICATIONS

$vi pod1.yml	$vi pod2.yml	$vi svc.yml
apiVersion: v1 kind: Pod metadata: name: pod1 labels: course: cka spec: containers: - name: c1 image: docker.io/openshift/hello-openshift ports: - containerPort: 8080	apiVersion: v1 kind: Pod metadata: name: pod2 labels: course: cka spec: containers: - name: c1 image: docker.io/httpd ports: - containerPort: 80	apiVersion: v1 kind: Service metadata: name: testing spec: selector: course: cka ports: - name: p1 protocol: TCP port: 80 targetPort: 80 - name: p3 protocol: TCP port: 8080 targetPort: 8080

Creating Pods

```
controlplane $ vi pod1.yml
controlplane $ kubectl create -f pod1.yml
pod/pod1 created
controlplane $ vi pod2.yml
controlplane $ kubectl create -f pod2.yml
pod/pod2 created
controlplane $ kubectl get pods -o wide --show-labels
NAME   READY   STATUS    RESTARTS   AGE   IP            NODE     NOMINATED NODE   READINESS GATES   LABELS
pod1   1/1     Running   0          39s   192.168.1.3   node01   <none>           <none>            course=cka
pod2   1/1     Running   0          11s   192.168.1.4   node01   <none>           <none>            course=cka
```

Creating the service

```
controlplane $ vi svc.yml
controlplane $ kubectl create -f svc.yml
service/testing created
controlplane $ kubectl get svc -o wide
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)           AGE   SELECTOR
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP           33d   <none>
testing      ClusterIP   10.110.145.143   <none>        8080/TCP,80/TCP   8s    course=cka
controlplane $ kubectl describe svc testing
Name:              testing
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          course=cka
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.110.145.143
IPs:               10.110.145.143
Port:              p1  8080/TCP
TargetPort:        8080/TCP
Endpoints:         192.168.1.3:8080,192.168.1.4:8080
Port:              p3  80/TCP
TargetPort:        80/TCP
Endpoints:         192.168.1.3:80,192.168.1.4:80
Session Affinity:  None
Events:            <none>
```

Output:

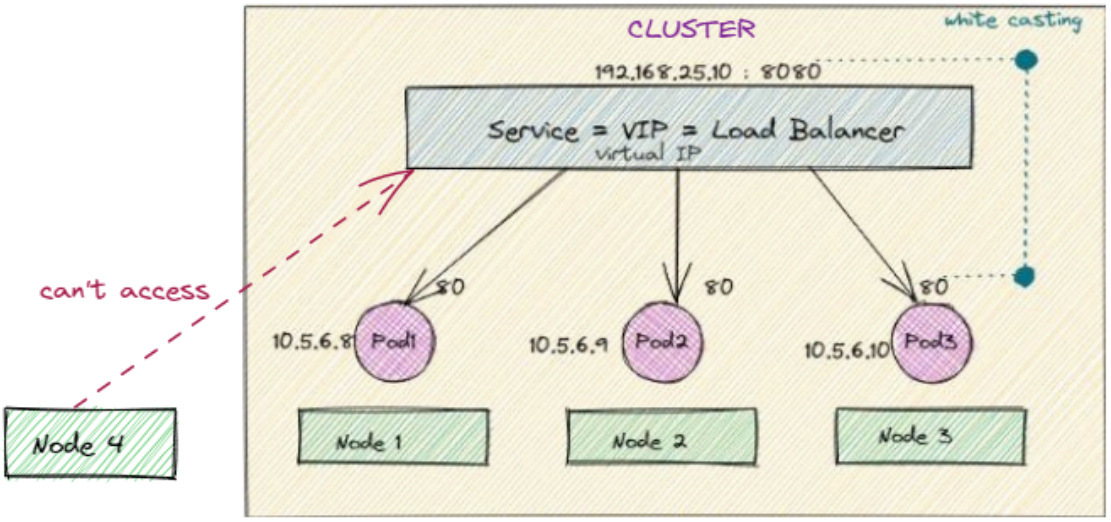
```
controlplane $ curl 10.110.145.143:80
<html><body><h1>It works!</h1></body></html>
controlplane $ curl 10.110.145.143:8080
Hello OpenShift!
```

Note:
Create a service even when we have a single pod.
This will help in many ways.

YONO SBI, Pokemon Go, mercedes benz, singapore airlines, UHG (UnitedHealth Group), spotify, etc are deployed on **Kubernetes**.

Problem Statement:

Service is a group of pods.
Service is only accessible inside the cluster (Service type 1: Cluster IP)
Therefore, the nodes outside the cluster can't access the service.

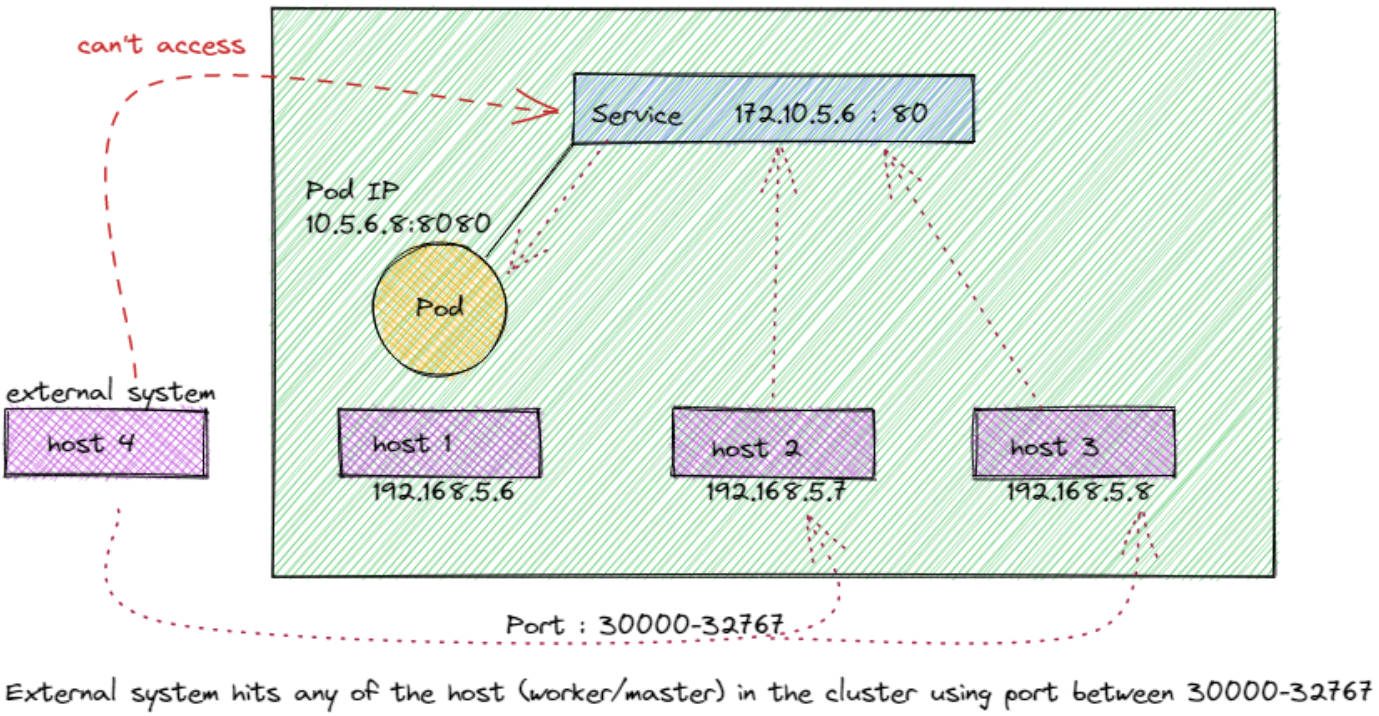


Here, comes the second Service Type:
2. NodePort

NodePort service allows the host / node outside the cluster to access the service inside the cluster.

Physical IP	Virtual IP / Logical IP
A physical IP address is a unique numerical identifier assigned to a network interface or device within a physical network.	A virtual IP address (VIP) is an IP address that is not associated with a specific physical network interface or device. Instead, it is assigned to a virtual resource or service that may run on multiple physical devices.
A physical IP address is directly associated with a specific physical device or network interface.	A virtual IP address is associated with a virtual resource or service that can run on multiple physical devices.
Host IP are physical Ip's	Service IP, Pod IP's are logical/virtual

Note:
In a **software-defined network (SDN)** environment, **service IP** addresses and **pod IP** addresses are typically **virtual or logical IP addresses**.
SDN = NAS (Network as a Service)



External system hits any of the host (worker/master) in the cluster using port between 30000-32767

Flow:

External system hits any of the host / nodes either it be master or worker.
The internal host takes the external host's request to the service.
The service takes it to the Pod.
And that's how the external host accesses the application in the pod in a cluster.

Practical:

Creating Pods

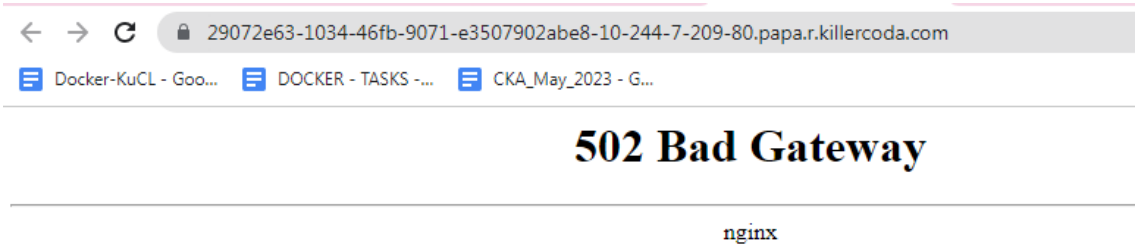
```
controlplane $ vi pod.yml
controlplane $ kubectl create -f pod.yml
pod/pod1 created
controlplane $ vi pod.yml
controlplane $ kubectl create -f pod.yml
pod/pod2 created
controlplane $ kubectl get pods -o wide --show-labels
NAME   READY   STATUS    RESTARTS   AGE   IP            NODE     NOMINATED NODE   READINESS GATES   LABELS
pod1   1/1     Running   0          20s   192.168.1.3   node01   <none>           <none>            course=cka
pod2   1/1     Running   0          10s   192.168.1.4   node01   <none>           <none>            course=cka
```

Creating a service

```
controlplane $ vi svc.yml
controlplane $ kubectl create -f svc.yml
service/svc1 created
```

```
controlplane $ kubectl describe svc svc1
Name:              svc1
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          course=cka
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.108.68.49
IPs:               10.108.68.49
Port:              p1  80/TCP
TargetPort:        80/TCP
Endpoints:         192.168.1.3:80,192.168.1.4:80
Session Affinity:  None
Events:            <none>
```

Accessing the application inside pods from External system / host



Error. Thus, can't access it!!

Editing the service => NodePort

```
controlplane $ kubectl edit svc testing
service/testing edited
controlplane $ kubectl describe svc testing
Name:                     testing
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 course=cka
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.102.173.214
IPs:                      10.102.173.214
Port:                     p1  80/TCP
TargetPort:               80/TCP
NodePort:                 p1  30402/TCP
Endpoints:                192.168.1.3:80,192.168.1.4:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

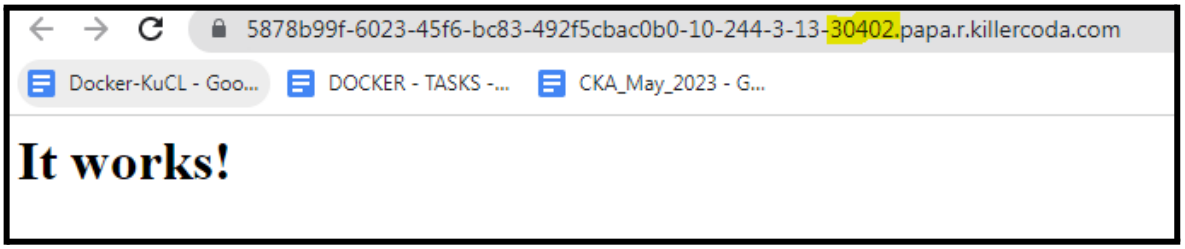
Editing inside svc file

```
  ports:
  - name: p1
    nodePort: 30402
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    course: cka
  sessionAffinity: None
  type: NodePort
status:
```

Accessing the service from nodes inside the cluster.

```
controlplane $ kubectl get nodes -o wide
NAME           STATUS   ROLES           AGE   VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION
  CONTAINER-RUNTIME
controlplane   Ready    control-plane   34d   v1.26.1   172.30.1.2    <none>        Ubuntu 20.04.5 LTS   5.4.0-131-generic
  containerd://1.6.12
node01         Ready    <none>          34d   v1.26.1   172.30.2.2    <none>        Ubuntu 20.04.5 LTS   5.4.0-131-generic
  containerd://1.6.12
controlplane $ curl 172.30.1.2:30402
<html><body><h1>It works!</h1></body></html>
controlplane $ curl 172.30.2.2:30402
<html><body><h1>It works!</h1></body></html>
```

Accessing the service from external host



Thus, the service is being accessed from an external host.

CREATING A POD MANUALLY USING YAML FILE

A. USING CREATE COMMAND

```
controlplane $ vi pod.yml
controlplane $ kubectl create -f pod.yml
pod/pod1 created
controlplane $ kubectl get pods
NAME   READY   STATUS    RESTARTS   AGE
pod1   1/1     Running   0          8s
```

But, the Create command is "**absolute**" i.e. we can't edit the pod after its creation.
If We need to change / edit something in the pod, either it be the label or the container port, etc;

```
controlplane $ vi pod.yml
controlplane $ kubectl create -f pod.yml
Error from server (AlreadyExists): error when creating "pod.yml": pods "pod1" already exists
```

We can't edit the pod. It throws out an error.

B. USING APPLY COMMAND

Apply command helps us to edit the pod whenever needed.

$vi podapply.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: pod1
  labels:
    state: maharashtra
spec:
  containers:
  - name: c1
    image: docker.io/httpd
    ports:
    - containerPort: 80
```

```
controlplane $ vi pod.yml
controlplane $ kubectl apply -f pod.yml
pod/pod1 created
controlplane $ kubectl get pods --show-labels
NAME   READY   STATUS              RESTARTS   AGE   LABELS
pod1   0/1     ContainerCreating   0          10s   state=maharashtra
controlplane $ kubectl get pods --show-labels
NAME   READY   STATUS    RESTARTS   AGE   LABELS
pod1   1/1     Running   0          17s   state=maharashtra
```

Pod is created.

If I want to change the label of the pod1, then we can directly edit the yaml file and use the apply command.

This will edit the label in the pod.

```
controlplane $ vi pod.yml
controlplane $ kubectl apply -f pod.yml
pod/pod1 configured
controlplane $ kubectl get pods --show-labels
NAME   READY   STATUS    RESTARTS   AGE   LABELS
pod1   1/1     Running   0          94s   state=gujarat
controlplane $ cat pod.yml
apiVersion: v1
kind: Pod
metadata:
  name: pod1
  labels:
    state: gujarat
spec:
  containers:
  - name: c1
    image: docker.io/httpd
    ports:
    - containerPort: 80
```

Now, let's try editing the image:

```
controlplane $ vi pod.yml
controlplane $ kubectl apply -f pod.yml
pod/pod1 configured
controlplane $ kubectl get pods --show-labels
NAME   READY   STATUS    RESTARTS      AGE     LABELS
pod1   1/1     Running   1 (5s ago)   4m24s   state=gujarat
controlplane $ cat pod.yml
apiVersion: v1
kind: Pod
metadata:
  name: pod1
  labels:
    state: gujarat
spec:
  containers:
  - name: c1
    image: docker.io/nginx
    ports:
    - containerPort: 80
```

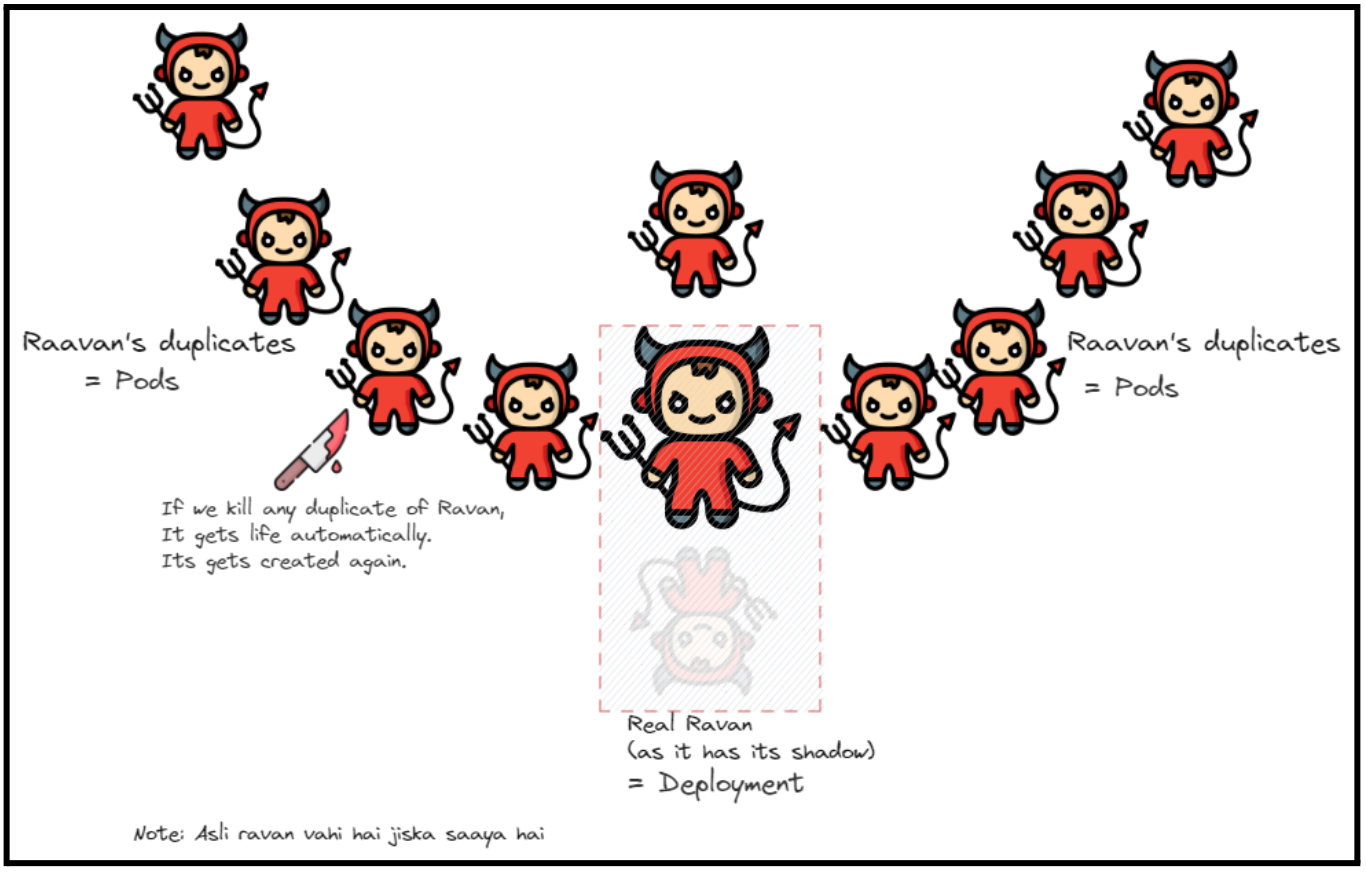
Note: We can't edit the port number

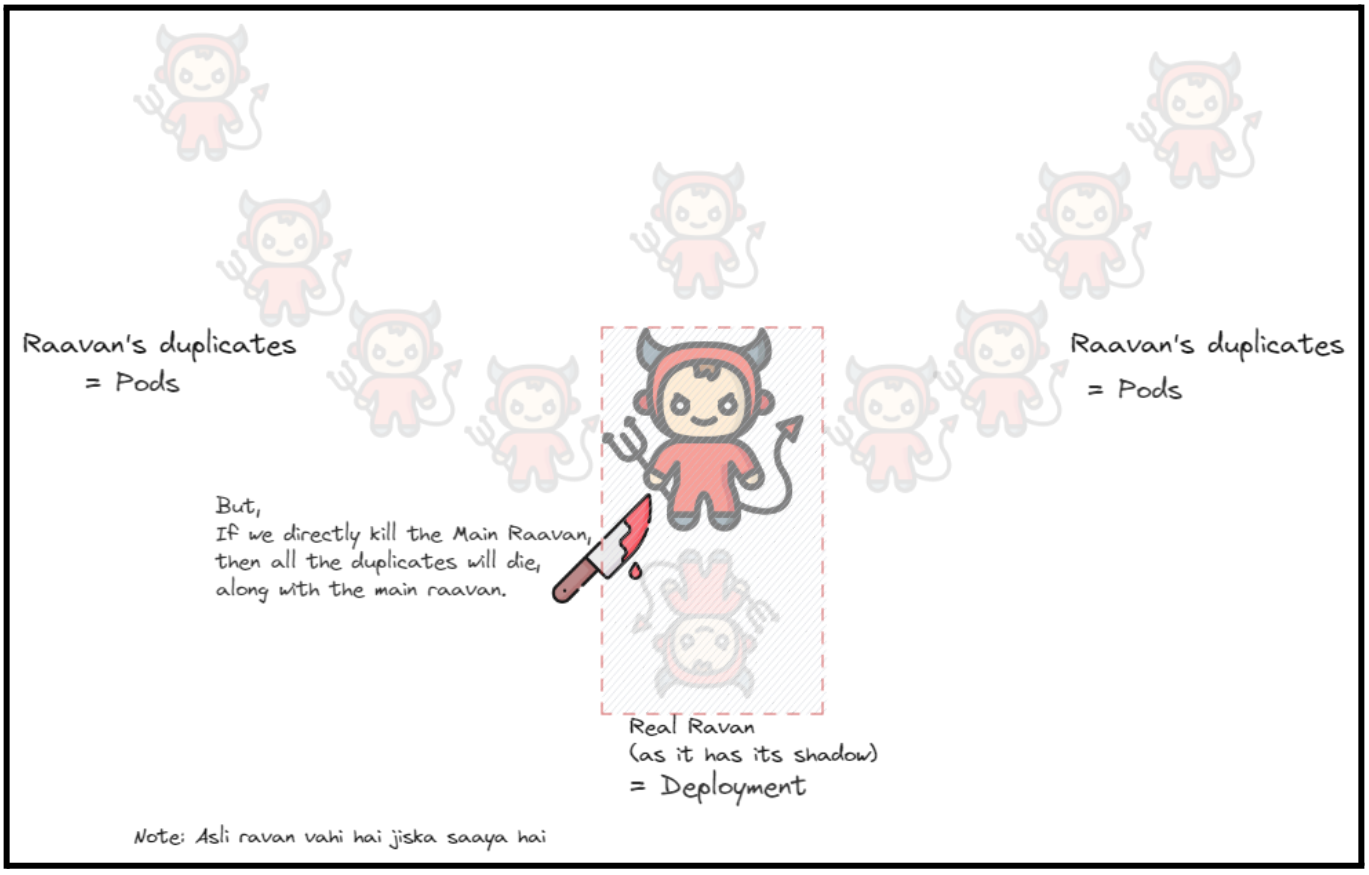
CREATING A POD WITHOUT YAML FILE

DEPLOYMENTS

For creating a pod without a yaml file, we use deployments.
Deployment is an object which helps in creating the pods.

Deployments are used for:
a. Creating and maintaining the pod.
b. Checking the health of the pod i.e. If the pod goes down, it brings the new pod up.
c. Updating an image


Raavan's duplicates
= Pods
Raavan's duplicates
= Pods
If we kill any duplicate of Ravan,
It gets life automatically.
Its gets created again.
Real Ravan
(as it has its shadow)
= Deployment
Note: Asli ravan vahi hai jiska saaya hai


Raavan's duplicates
= Pods
Raavan's duplicates
= Pods
But,
If we directly kill the Main Raavan,
then all the duplicates will die,
along with the main raavan.
Real Ravan
(as it has its shadow)
= Deployment
Note: Asli ravan vahi hai jiska saaya hai

Creating Deployment (2 ways)

a. Using CLI

```
controlplane $ kubectl create deployment d1 --image docker.io/httpd
deployment.apps/d1 created
controlplane $ kubectl get deployments
NAME   READY   UP-TO-DATE   AVAILABLE   AGE
d1     0/1     1            0           5s
controlplane $ kubectl get deployments -w
NAME   READY   UP-TO-DATE   AVAILABLE   AGE
d1     1/1     1            1           10s
controlplane $ kubectl get pods
NAME                  READY   STATUS    RESTARTS   AGE
d1-76ddc565c7-qs5kp   1/1     Running   0          26s
controlplane $ kubectl get pods --show-labels
NAME                  READY   STATUS    RESTARTS   AGE   LABELS
d1-76ddc565c7-qs5kp   1/1     Running   0          32s   app=d1,pod-template-hash=76ddc565c7
```

Therefore, creating a deployment results in creation of pods.

Deleting the pod created by deployment:

```
controlplane $ kubectl get pods
NAME                  READY   STATUS    RESTARTS   AGE
d1-76ddc565c7-qs5kp   1/1     Running   0          94s
controlplane $ kubectl delete pod d1-76ddc565c7-qs5kp
pod "d1-76ddc565c7-qs5kp" deleted
controlplane $ kubectl get pods
NAME                  READY   STATUS              RESTARTS   AGE
d1-76ddc565c7-9x72s   0/1     ContainerCreating   0          11s
controlplane $ kubectl get pods
NAME                  READY   STATUS              RESTARTS   AGE
d1-76ddc565c7-9x72s   0/1     ContainerCreating   0          15s
controlplane $ kubectl get pods -w
NAME                  READY   STATUS    RESTARTS   AGE
d1-76ddc565c7-9x72s   1/1     Running   0          19s
^Ccontrolplane $ kubectl get pods --show-labels
NAME                  READY   STATUS    RESTARTS   AGE   LABELS
d1-76ddc565c7-9x72s   1/1     Running   0          29s   app=d1,pod-template-hash=76ddc565c7
```

Pods get created automatically.
Therefore, deployments constantly check the health of the pod. If any pod goes down, it brings the new pod up.
Deployment maintains the number of pods in it at any cost.

Creating **Multiple replicas** of pods in Deployment.

```
controlplane $ kubectl create deployment d2 --image=docker.io/openshift/hello-openshift --replicas=5
deployment.apps/d2 created
controlplane $ kubectl get deployments.apps -w
NAME   READY   UP-TO-DATE   AVAILABLE   AGE
d1     1/1     1            1           5m58s
d2     4/5     5            4           8s
d2     5/5     5            5           9s
^Ccontrolplane $ kubectl get pods --show-labels -w
NAME                  READY   STATUS    RESTARTS   AGE     LABELS
d1-76ddc565c7-9x72s   1/1     Running   0          4m33s   app=d1,pod-template-hash=76ddc565c7
d2-86f7b9599-685bp    1/1     Running   0          29s     app=d2,pod-template-hash=86f7b9599
d2-86f7b9599-dz2kq    1/1     Running   0          29s     app=d2,pod-template-hash=86f7b9599
d2-86f7b9599-jw95t    1/1     Running   0          29s     app=d2,pod-template-hash=86f7b9599
d2-86f7b9599-nghlw    1/1     Running   0          29s     app=d2,pod-template-hash=86f7b9599
d2-86f7b9599-qc68z    1/1     Running   0          29s     app=d2,pod-template-hash=86f7b9599
```

Deployment creates multiple **identical pods**.

Deleting 2 pods:

```
controlplane $ kubectl get pods --show-labels
NAME                  READY   STATUS    RESTARTS   AGE     LABELS
d1-76ddc565c7-9x72s   1/1     Running   0          6m44s   app=d1,pod-template-hash=76ddc565c7
d2-86f7b9599-685bp    1/1     Running   0          2m40s   app=d2,pod-template-hash=86f7b9599
d2-86f7b9599-dz2kq    1/1     Running   0          2m40s   app=d2,pod-template-hash=86f7b9599
d2-86f7b9599-jw95t    1/1     Running   0          2m40s   app=d2,pod-template-hash=86f7b9599
d2-86f7b9599-nghlw    1/1     Running   0          2m40s   app=d2,pod-template-hash=86f7b9599
d2-86f7b9599-qc68z    1/1     Running   0          2m40s   app=d2,pod-template-hash=86f7b9599
controlplane $ kubectl delete pod d2-86f7b9599-qc68z d2-86f7b9599-nghlw
pod "d2-86f7b9599-qc68z" deleted
pod "d2-86f7b9599-nghlw" deleted
controlplane $ kubectl get pods --show-labels
NAME                  READY   STATUS    RESTARTS   AGE     LABELS
d1-76ddc565c7-9x72s   1/1     Running   0          7m9s    app=d1,pod-template-hash=76ddc565c7
d2-86f7b9599-4x6q8    1/1     Running   0          3s      app=d2,pod-template-hash=86f7b9599
d2-86f7b9599-685bp    1/1     Running   0          3m5s    app=d2,pod-template-hash=86f7b9599
d2-86f7b9599-dz2kq    1/1     Running   0          3m5s    app=d2,pod-template-hash=86f7b9599
d2-86f7b9599-jw95t    1/1     Running   0          3m5s    app=d2,pod-template-hash=86f7b9599
d2-86f7b9599-ng4dx    1/1     Running   0          3s      app=d2,pod-template-hash=86f7b9599
```

Pods get created automatically. {**High Availability**}

Deployment creates a **sub-object** in the backend: **REPLICA SET**, for maintaining the number of replicas.

Listing the Replica sets:

```
controlplane $ kubectl get rs
NAME            DESIRED   CURRENT   READY   AGE
d1-76ddc565c7   1         1         1       11m
d2-86f7b9599    5         5         5       5m38s
```

Creating a Service of the Deployment

Deployment: d1

Expose command is used to expose the deployment to service.

```
controlplane $ kubectl get deployments
NAME   READY   UP-TO-DATE   AVAILABLE   AGE
d1     1/1     1            1           12m
d2     5/5     5            5           6m40s
controlplane $ kubectl expose deployment d1 --port=80 --target-port=80
service/d1 exposed
controlplane $ kubectl get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
d1           ClusterIP   10.99.5.211    <none>        80/TCP    5s
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP   2d12h
```

We need to specify:

Port: through which service will be accessed : [Source]

Target-Port: through which pods [containers] will be accessed : [Destination]

If we do not specify target-port value, then it takes the value from - -port.

Type: Service type.

By default, it takes **Cluster IP,** which is the first type of service.

Describing Service:

```
controlplane $ kubectl describe service d1
Name:              d1
Namespace:         default
Labels:            app=d1
Annotations:       <none>
Selector:          app=d1
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.99.5.211
IPs:               10.99.5.211
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         192.168.1.5:80
Session Affinity:  None
Events:            <none>
```

Output:

```
controlplane $ kubectl get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
d1           ClusterIP   10.99.5.211    <none>        80/TCP    6m
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP   2d12h
controlplane $ curl 10.99.5.211:80
<html><body><h1>It works!</h1></body></html>
```

Deployment d2

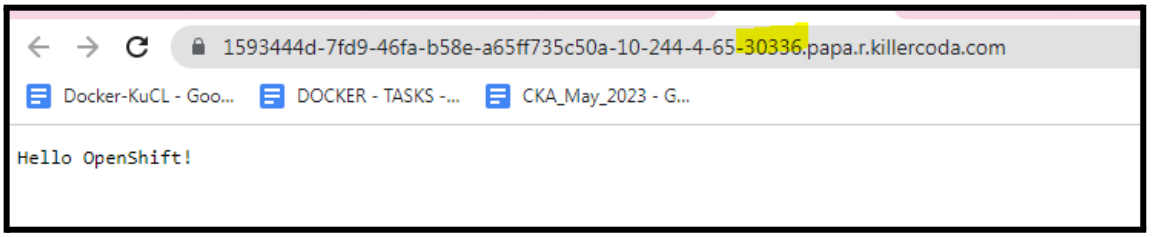
Exposing the deployment to service:

Not mentioning the --target-port. It will take the port value the same as --port.

```
controlplane $ kubectl expose deployment d2 --port=8080 --type=NodePort
service/d2 exposed
controlplane $ kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
d1           ClusterIP   10.99.5.211     <none>        80/TCP           7m53s
d2           NodePort    10.111.91.245   <none>        8080:30336/TCP   4s
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP          2d12h
controlplane $ kubectl describe svc d2
Name:                     d2
Namespace:                default
Labels:                   app=d2
Annotations:              <none>
Selector:                 app=d2
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.111.91.245
IPs:                      10.111.91.245
Port:                     <unset>  8080/TCP
TargetPort:               8080/TCP
NodePort:                 <unset>  30336/TCP
Endpoints:                192.168.0.10:8080,192.168.0.8:8080,192.168.1.7:8080 + 2 more..
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

Output:

```
controlplane $ kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
d1           ClusterIP   10.99.5.211     <none>        80/TCP           9m41s
d2           NodePort    10.111.91.245   <none>        8080:30336/TCP   112s
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP          2d12h
controlplane $ curl localhost:30336
Hello OpenShift!
```



b. Using YAML file

```
controlplane $ kubectl create deployment d1 --image=docker.io/httpd --dry-run=client -o yaml > d1.yml
controlplane $ cat d1.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: d1
  name: d1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: d1
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: d1
    spec:
      containers:
      - image: docker.io/httpd
        name: httpd
        resources: {}
status: {}
```

--dry-run : for testing purpose
-o : output
yaml : format

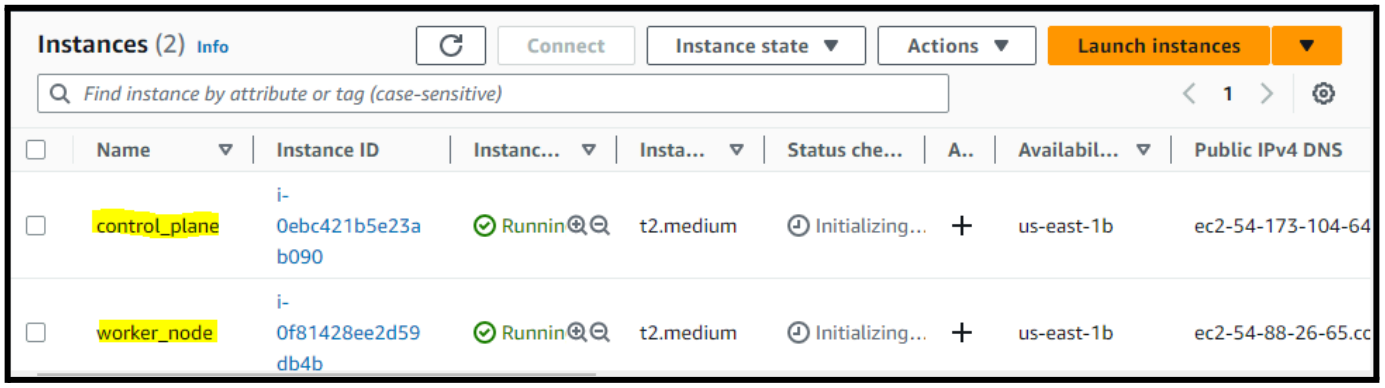
[Remaining till k8s cluster]

CREATING A KUBERNETES CLUSTER FROM SCRATCH

Using Debian : Ubuntu machines

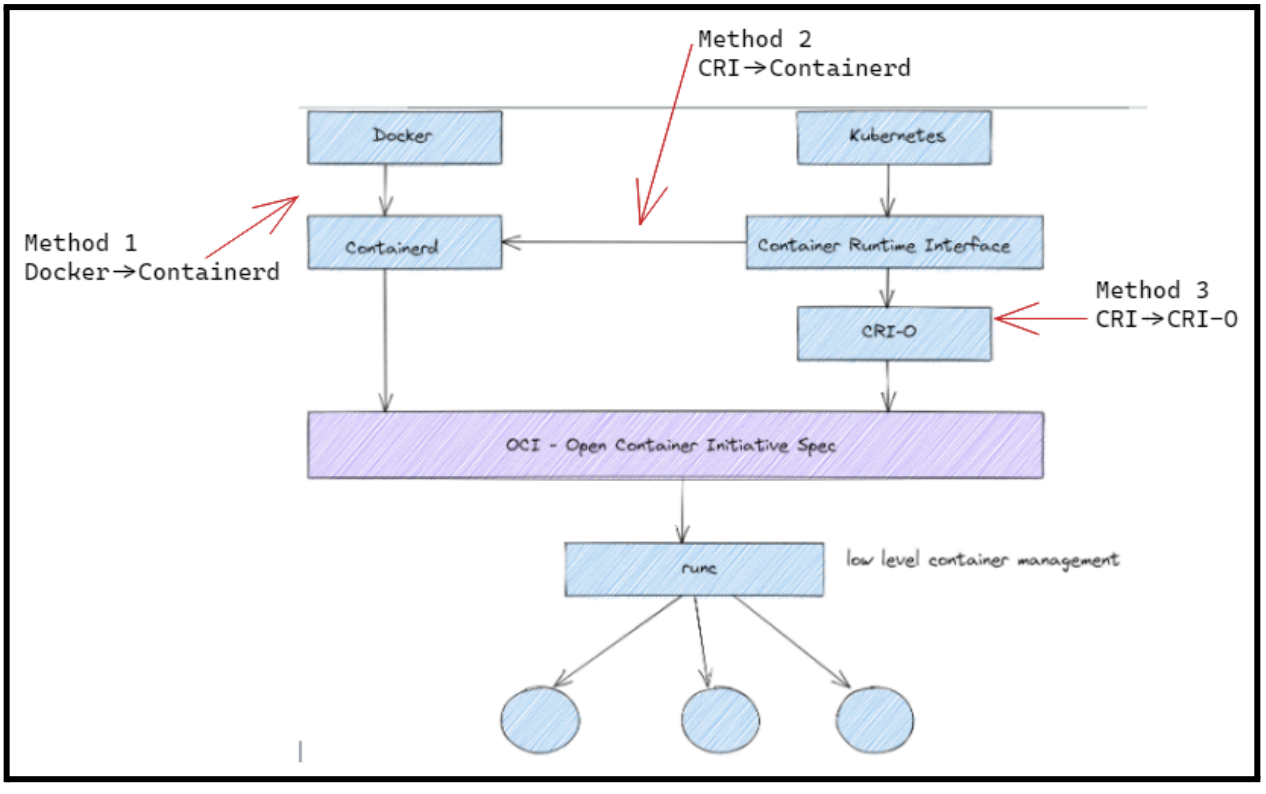
Prerequisites:
- One or more machines running a **deb-compatible** Linux OS.
- **2 GiB** or more of **RAM** per machine.
- At least **2 CPU**s on the machine that we will use as a control-plane node.
- Full **network connectivity** among all machines in the cluster.

Using AWS Instances (Ubuntu machines)



Both the ubuntu instances should be in the same vpc.
Instance type = t2.medium
Now, these instances are my hosts / nodes.

Container Runtimes



Here, we will be going with the **method 1: docker -> containerd**

PREPARING THE HOSTS

We need to install a container runtime into each node in the cluster so that Pods can run there. Kubernetes 1.27 requires that you use a runtime that conforms with the Container Runtime Interface (CRI).

On Control-plane: Master

1. Enabling IPV4 Forwarding and let iptables see bridged traffic

```
ubuntu@ip-172-31-90-248:~$ history
    1  cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF

    2  sudo modprobe overlay
    3  sudo modprobe br_netfilter
    4  # sysctl params required by setup, params persist across reboots
    5  cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
EOF

    6  # Apply sysctl params without reboot
    7  sudo sysctl --system
    8  history
```

2. Installing Docker

```
 12  curl -fsSL https://get.docker.com -o get-docker.sh
 13  sudo sh ./get-docker.sh
 14  sudo systemctl start docker
 15  sudo systemctl status docker
 16  sudo systemctl enable docker
 17  sudo docker --v
 18  sudo docker -v
 19  sudo docker ps
```

3. Enable the containerd plugin and restart docker and containerd service

```
ubuntu@ip-172-31-90-248:~$ sudo vi /etc/containerd/config.toml
ubuntu@ip-172-31-90-248:~$ ubuntu@ip-172-31-90-248:~$
ubuntu@ip-172-31-90-248:~$ cat /etc/containerd/config.toml | grep enable_plugins
enable_plugins = ["containerd"]
ubuntu@ip-172-31-90-248:~$ sudo systemctl restart containerd
ubuntu@ip-172-31-90-248:~$ sudo systemctl restart docker
ubuntu@ip-172-31-90-248:~$
```

4. Installing kubeadm, kubelet and kubectl

```
 25  sudo apt-get update
 26  sudo apt-get install -y apt-transport-https ca-certificates curl
 27  curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo gpg --dearmor -o /etc/apt/keyrings/kubern
tes-archive-keyring.gpg
 28  echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xeni
l main" | sudo tee /etc/apt/sources.list.d/kubernetes.list
 29  sudo apt-get update
 30  sudo apt-get install -y kubelet kubeadm kubectl
```

Therefore, **Kubernetes is now installed o**n our Control-plane.

Creating this instance / node as my **control-plane /** master/

```
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rotatable kubelet client certificate and ke
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 172.31.90.248:6443 --token 3urdop.r25zjlpt6xijiso1 \
```

Creating the K8s cluster

```
 34  sudo kubeadm init
 35  kubectl get nodes
 36  mkdir -p $HOME/.kube
 37  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
 38  sudo chown $(id -u):$(id -g) $HOME/.kube/config
 39  kubectl get nodes
```

Cluster nodes:

```
ubuntu@ip-172-31-90-248:~$ kubectl get nodes
NAME               STATUS     ROLES           AGE    VERSION
ip-172-31-90-248   NotReady   control-plane   3m4s   v1.27.2
ubuntu@ip-172-31-90-248:~$
```

Doing the same procedure on the **worker node**, and adding it as a worker to the cluster.
For this just need to copy the **token** created on the worker node.

```
ubuntu@worker: $ sudo kubeadm join 172.31.17.193:6443 --token 918l31.7y90ltidwj22j9bg --discovery-token-ca-cert-hash sha256:7219a90ead
c09d8204f78c6aa465ad77986203bcceddc74842d084d5829b8a28
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

On **control-plane:**

```
ubuntu@controlplane:~$ kubectl get nodes
NAME           STATUS     ROLES           AGE     VERSION
controlplane   NotReady   control-plane   3m20s   v1.27.2
worker         NotReady   <none>          110s    v1.27.2
```

Status: Not Healthy.
Therefore, we need an overlay so that the nodes talk with each other.

Installing Addons: **Weave Net** : CNI Plugins.
CNI-Plugins is the overlay network : so as to connect the nodes / hosts

```
ubuntu@controlplane: $ kubectl apply -f https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s.y
aml
serviceaccount/weave-net unchanged
clusterrole.rbac.authorization.k8s.io/weave-net unchanged
clusterrolebinding.rbac.authorization.k8s.io/weave-net unchanged
role.rbac.authorization.k8s.io/weave-net unchanged
rolebinding.rbac.authorization.k8s.io/weave-net unchanged
daemonset.apps/weave-net configured
ubuntu@controlplane: $ kubectl get nodes
NAME           STATUS     ROLES           AGE     VERSION
controlplane   NotReady   control-plane   5m9s    v1.27.2
worker         NotReady   <none>          3m39s   v1.27.2
ubuntu@controlplane: $ kubectl get nodes -w
NAME           STATUS     ROLES           AGE     VERSION
controlplane   NotReady   control-plane   5m12s   v1.27.2
worker         NotReady   <none>          3m42s   v1.27.2
controlplane   NotReady   control-plane   5m13s   v1.27.2

controlplane   NotReady   control-plane   6m9s    v1.27.2
worker         NotReady   <none>          4m39s   v1.27.2
^Cubuntu@controlplane:~$ kubectl get nodes -w
NAME           STATUS     ROLES           AGE     VERSION
controlplane   NotReady   control-plane   6m12s   v1.27.2
worker         NotReady   <none>          4m42s   v1.27.2
worker         Ready      <none>          4m44s   v1.27.2
worker         Ready      <none>          4m44s   v1.27.2
controlplane   Ready      control-plane   6m15s   v1.27.2
controlplane   Ready      control-plane   6m15s   v1.27.2
```

Therefore, the K8s cluster is Ready.

==========

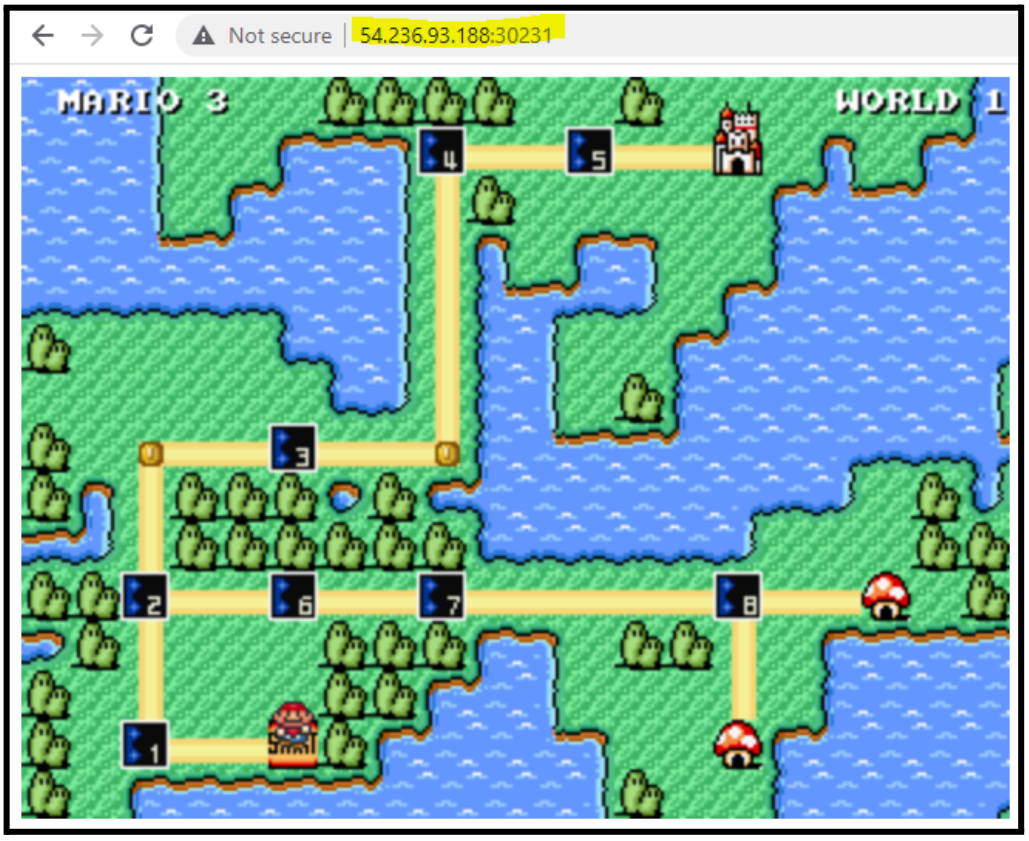
Creating Deployments:

Task 11:

Deploy the k8s cluster on Ubuntu and deploy mario using nodeport

```
ubuntu@controlplane:~$ kubectl create deployment d1 --image=docker.io/pengbai/docker-supermari
o
deployment.apps/d1 created
ubuntu@controlplane:~$ kubectl get deployments
NAME   READY   UP-TO-DATE   AVAILABLE   AGE
d1     0/1     1            0           7s
ubuntu@controlplane:~$ kubectl get deployments -w
NAME   READY   UP-TO-DATE   AVAILABLE   AGE
d1     0/1     1            0           11s
d1     1/1     1            1           13s
^Cubuntu@controlplane:~$ kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
d1-7c5bf76ffb-7xrnz   1/1     Running   0          20s
ubuntu@controlplane:~$ kubectl expose deployment d1 --port=8080 --type=NodePort
service/d1 exposed
ubuntu@controlplane:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
d1           NodePort    10.110.0.166   <none>        8080:30231/TCP   5s
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP          3m56s
ubuntu@controlplane:~$ kubectl describe svc d1
Name:                     d1
Namespace:                default
Labels:                   app=d1
Annotations:              <none>
Selector:                 app=d1
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.110.0.166
IPs:                      10.110.0.166
Port:                     <unset>  8080/TCP
TargetPort:               8080/TCP
NodePort:                 <unset>  30231/TCP
Endpoints:                10.44.0.3:8080
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

Output



==========

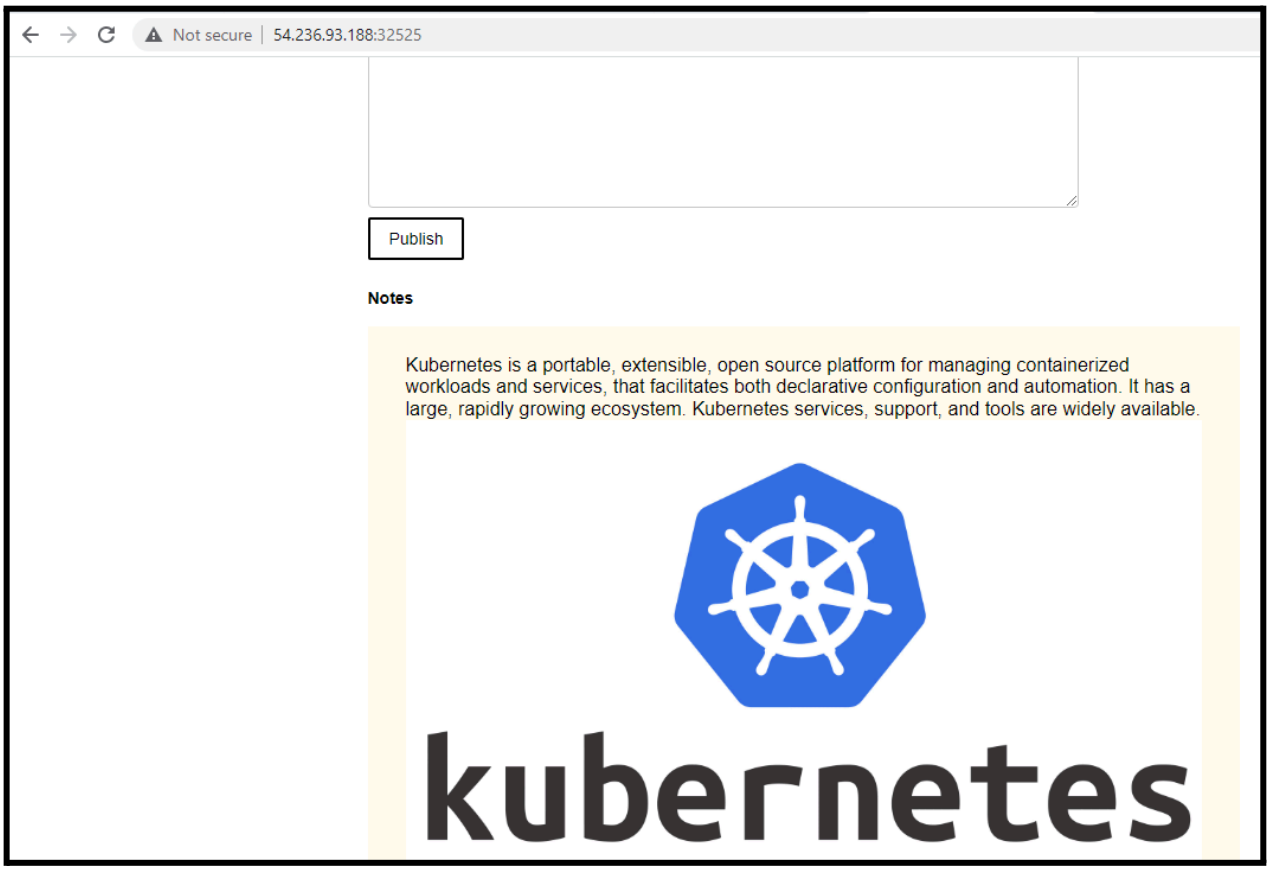
Task 13

Deploy the Multitier Application on k8s cluster created on ubuntu machines.

Deploying "**knote**" multi-tier application: https://github.com/ashutoshbhakare/knote

```
ubuntu@controlplane:~$ ls
get-docker.sh  knote
ubuntu@controlplane:~$ cd knote/
ubuntu@controlplane:~/knote$ ls
dbdeployment.yml  dbsvc.yml  knotedeployment.yml  knotesvc.yml  noteforpvpvc
ubuntu@controlplane:~/knote$ kubectl create -f .
deployment.apps/mongo created
service/mongo created
deployment.apps/knote created
service/knote created
ubuntu@controlplane:~/knote$ kubectl get deployments
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
d1      1/1     1            1           14m
knote   0/1     1            0           8s
mongo   0/1     1            0           8s
ubuntu@controlplane:~/knote$ kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
d1           NodePort    10.110.0.166     <none>        8080:30231/TCP   14m
knote        NodePort    10.100.2.103     <none>        80:32525/TCP     13s
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP          18m
mongo        ClusterIP   10.107.162.125   <none>        27017/TCP        13s
ubuntu@controlplane:~/knote$ kubectl get pods
NAME                     READY   STATUS              RESTARTS   AGE
d1-7c5bf76ffb-7xrnz      1/1     Running             0          15m
knote-65f678987c-qtttp   0/1     ContainerCreating   0          17s
mongo-598c78777c-dgj9l   1/1     Running             0          17s
ubuntu@controlplane:~/knote$
```

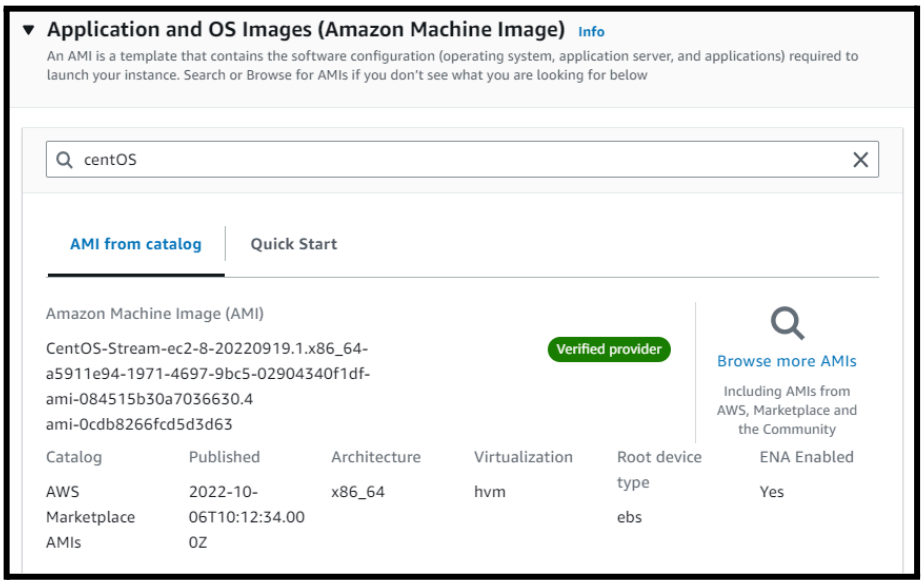
Output:


Not secure | 54.236.93.188:32525
Publish
Notes
Kubernetes is a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.
kubernetes

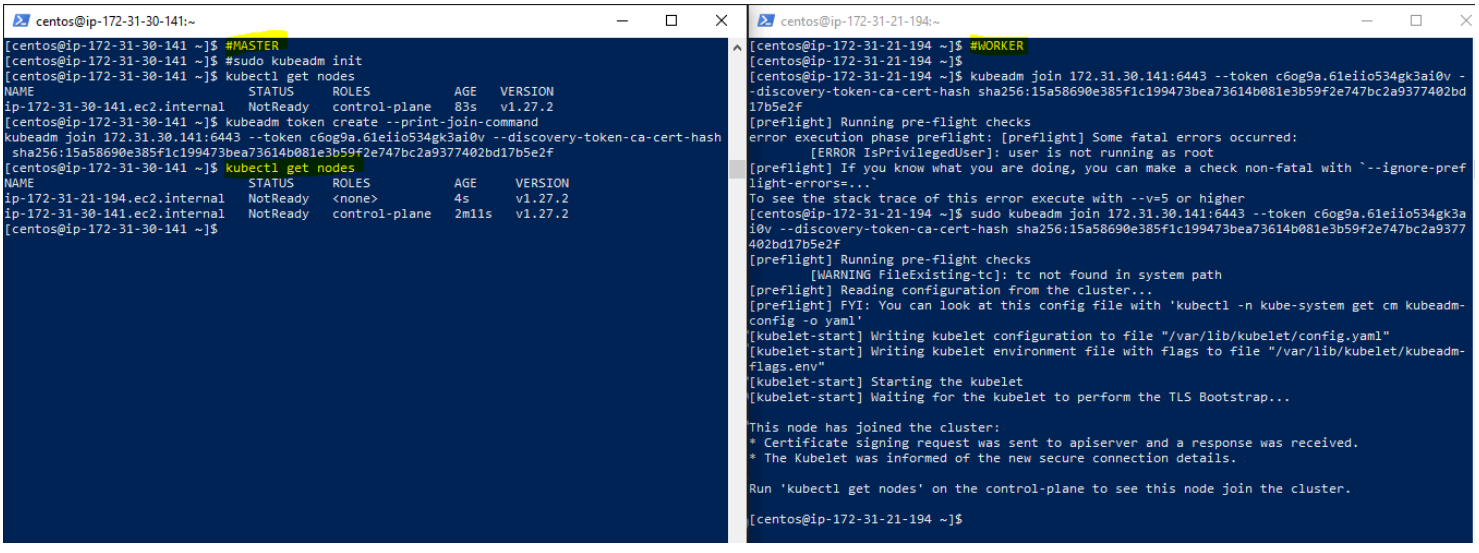
Task 12

Deploy the k8s cluster on Centos and deploy mario using nodeport.

Launching 2 CentOS Instances on AWS



Preparing both the machines.



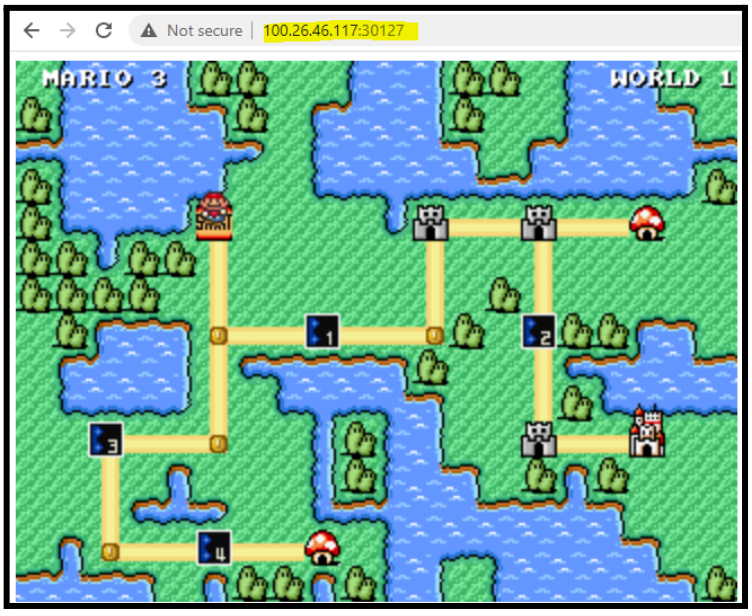
```
[centos@ip-172-31-30-141 ~]$ kubectl apply -f https://github.com/weaveworks/weave/releases/dow
nload/v2.8.1/weave-daemonset-k8s.yaml
serviceaccount/weave-net created
clusterrole.rbac.authorization.k8s.io/weave-net created
clusterrolebinding.rbac.authorization.k8s.io/weave-net created
role.rbac.authorization.k8s.io/weave-net created
rolebinding.rbac.authorization.k8s.io/weave-net created
daemonset.apps/weave-net created
[centos@ip-172-31-30-141 ~]$ kubectl get nodes
NAME                          STATUS     ROLES           AGE     VERSION
ip-172-31-21-194.ec2.internal   NotReady   <none>          2m4s    v1.27.2
ip-172-31-30-141.ec2.internal   NotReady   control-plane   4m11s   v1.27.2
[centos@ip-172-31-30-141 ~]$ kubectl get nodes
NAME                          STATUS     ROLES           AGE     VERSION
ip-172-31-21-194.ec2.internal   Ready      <none>          2m14s   v1.27.2
ip-172-31-30-141.ec2.internal   NotReady   control-plane   4m21s   v1.27.2
[centos@ip-172-31-30-141 ~]$ kubectl get nodes
NAME                          STATUS   ROLES           AGE     VERSION
ip-172-31-21-194.ec2.internal   Ready    <none>          2m22s   v1.27.2
ip-172-31-30-141.ec2.internal   Ready    control-plane   4m29s   v1.27.2
```

Thus, both the nodes are Ready.

Let’s create a Mario Deployment & service

```
centos@ip-172-31-30-141:~
[centos@ip-172-31-30-141 ~]$ kubectl create deployment d1 --image=docker.io/pengbai/docker-sup
ermario
deployment.apps/d1 created
[centos@ip-172-31-30-141 ~]$ kubectl get deployments
NAME   READY   UP-TO-DATE   AVAILABLE   AGE
d1     0/1     1            0           6s
[centos@ip-172-31-30-141 ~]$ kubectl get pods -w
NAME                  READY   STATUS              RESTARTS   AGE
d1-7c5bf76ffb-ppr9z   0/1     ContainerCreating   0          11s
d1-7c5bf76ffb-ppr9z   1/1     Running             0          15s
^C[centos@ip-172-31-30-141 ~]$ kubectl expose deployment d1 --port=8080 --type=NodePort
service/d1 exposed
[centos@ip-172-31-30-141 ~]$ kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
d1           NodePort    10.111.74.215   <none>        8080:30127/TCP   4s
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP          6m23s
[centos@ip-172-31-30-141 ~]$
```

Output



KUBERNETES DASHBOARD

Kubernetes can be accessed via:

a. CLI
b. API call
c. **K8s Dashboard**

Kubernetes Dashboard is a web-based user interface that provides a graphical interface for managing and monitoring Kubernetes clusters. It allows users to view and interact with various resources, such as pods, services, deployments, and namespaces, within the cluster.

K8s Deployment will create the K8s Dashboard.
The service of deployment will be created, and this will allow the other outside user to access the dashboard.

Deploying a K8s Dashboard

Prerequisite: A K8s Cluster : Master node and a Worker node.
[I have Schedulable master, no worker node]

Creating dashboard through YAML file:
[Apply: can support the URL of yaml file]
Link: kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.7.0/aio/deploy/recommended.yaml

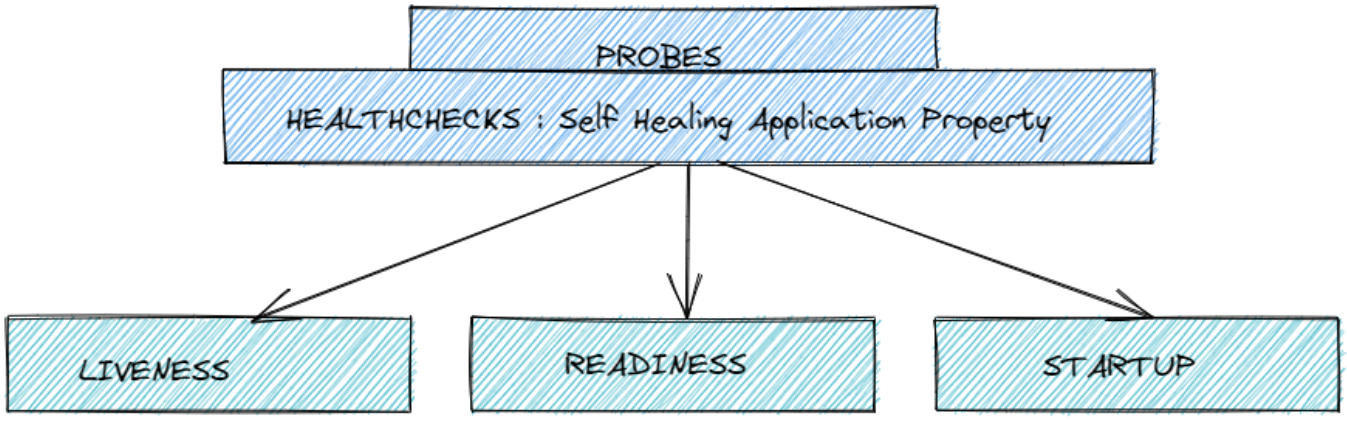
ROLE BASED ACCESS CONTROL (remaining)

SELF HEALING APPLICATION PROPERTY

=> Healthcheck
T checks whether the container is Healthy or NOT?
If container is Healthy ⇒ Okay
If container UNHEALTHY ⇒ Restart the container

NOTE:
INIT container runs **before** the application container.
Healthcheck runs after the application container is started. It runs on the application container to check its health.

We can have **both** Init and health check containers in one pod.yml file.



Timeout, InitialDelaySeconds and PeriodSec => common in all the 3 types

LIVENESS PROBE

- Checks whether the application is Live or Not.
- If not, then restart the application container.
- The events won't be available in "logs", it will be available in "describe"; because it is not the log of the application.

The kubelet uses liveness probes to know when to restart a container. For example, liveness probes could catch a deadlock, where an application is running, but unable to make progress. Restarting a container in such a state can help to make the application more available despite bugs.

Liveness Probe Example 1:
pod.yml

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec
spec:
  containers:
```

```
- name: liveness
  image: registry.k8s.io/busybox
  args:
  - /bin/sh
  - -c
  - touch /tmp/healthy; sleep 30; rm -f /tmp/healthy; sleep 600
  livenessProbe:
    exec:
      command:
      - cat
      - /tmp/healthy
    initialDelaySeconds: 5
    periodSeconds: 5
```

Here,
File gets created: /tmp/healthy
Delay of 30 seconds
Command will be executed: cat /tmp/healthy **Healthy**
Remove the file: /tmp/healthy
Delay of 600 seconds == 10 minutes
Command will be executed: cat /tmp/healthy **Unhealthy**
Therefore, now the container will restart

Creating the pod

```
controlplane $ vi pod.yml
controlplane $ kubectl apply -f pod.yml
pod/liveness-exec created
controlplane $ kubectl get pods -w
NAME            READY   STATUS              RESTARTS   AGE
liveness-exec   0/1     ContainerCreating   0          10s
liveness-exec   1/1     Running             0          17s
```

$**kubectl describe pod liveness-exec**

```
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  25s   default-scheduler  Successfully assigned default/liveness-exec to node01
  Normal  Pulling    23s   kubelet            Pulling image "registry.k8s.io/busybox"
  Normal  Pulled     11s   kubelet            Successfully pulled image "registry.k8s.io/busybox" in 11.475137153s (11.475144038s including waiting
)
  Normal  Created    11s   kubelet            Created container liveness
  Normal  Started    9s    kubelet            Started container liveness
```

Describe again

```
Events:
  Type     Reason     Age                From               Message
  ----     ------     ----               ----               -------
  Normal   Scheduled  92s                default-scheduler  Successfully assigned default/liveness-exec to node01
  Normal   Pulled     77s                kubelet            Successfully pulled image "registry.k8s.io/busybox" in 11.475137153s (11.475144038s inc
luding waiting)
  Warning  Unhealthy  36s (x3 over 46s)  kubelet            Liveness probe failed: cat: can't open '/tmp/healthy': No such file or directory
  Normal   Killing    36s                kubelet            Container liveness failed liveness probe, will be restarted
  Normal   Pulling    6s (x2 over 89s)   kubelet            Pulling image "registry.k8s.io/busybox"
  Normal   Created    3s (x2 over 77s)   kubelet            Created container liveness
  Normal   Started    3s (x2 over 75s)   kubelet            Started container liveness
  Normal   Pulled     3s                 kubelet            Successfully pulled image "registry.k8s.io/busybox" in 3.081461116s (3.081467293s inclu
```

Container will restart now.

```
controlplane $ kubectl get pods -w
NAME            READY   STATUS    RESTARTS      AGE
liveness-exec   1/1     Running   4 (35s ago)   5m46s
```

Liveness Probe Example 2
Pod1.yml

```
cat custom.yml
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec1
spec:
  containers:
  - name: liveness
    image: docker.io/httpd
    livenessProbe:
      exec:
        command:
        - grep
        - -i
        - 'works'
        - /usr/local/apache2/htdocs/index.html
      initialDelaySeconds: 5
      periodSeconds: 5
```

```
controlplane $ kubectl apply -f pod1.yml
pod/liveness-exec1 created
controlplane $ kubectl get pods -w
NAME             READY   STATUS              RESTARTS   AGE
liveness-exec1   0/1     ContainerCreating   0          10s
liveness-exec1   1/1     Running             0          11s
^Ccontrolplane $ kubectl exec -it liveness-exec1
error: you must specify at least one command for the container
controlplane $ kubectl exec -it liveness-exec1 bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
root@liveness-exec1:/usr/local/apache2# cat htdocs/index.html
<html><body><h1>It works!</h1></body></html>
root@liveness-exec1:/usr/local/apache2# exit
exit
```

Here, index.html has “It works”, therefore there is no error, and the pod is in healthy condition.

```
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  85s   default-scheduler  Successfully assigned default/liveness-exec1 to node01
  Normal  Pulling    82s   kubelet            Pulling image "docker.io/httpd"
  Normal  Pulled     74s   kubelet            Successfully pulled image "docker.io/httpd" in 7.368889234s (7.368900705s including waiting)
  Normal  Created    74s   kubelet            Created container liveness
  Normal  Started    74s   kubelet            Started container liveness
```

But if we change the content in the index.html file?

```
controlplane $ kubectl exec -it liveness-exec1 bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
root@liveness-exec1:/usr/local/apache2# echo "Hello World" > htdocs/index.html
root@liveness-exec1:/usr/local/apache2# cat htdocs/index.html
Hello World
root@liveness-exec1:/usr/local/apache2# exit
exit
```

Now, it won't show the result when the command "grep -i 'works' /usr/local/apache2/htdocs/index.html is executed.
Therefore, the container will become Unhealthy, and thus restart.

```
                    node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type     Reason     Age                 From               Message
  ----     ------     ----                ----               -------
  Normal   Scheduled  8m11s               default-scheduler  Successfully assigned default/liveness-exec1 to node01
  Normal   Pulled     8m                  kubelet            Successfully pulled image "docker.io/httpd" in 7.368889234s (7.368900705s including wai
ting)
  Warning  Unhealthy  10s (x3 over 20s)   kubelet            Liveness probe failed:
  Normal   Killing    10s                 kubelet            Container liveness failed liveness probe, will be restarted
  Normal   Pulling    9s (x2 over 8m8s)   kubelet            Pulling image "docker.io/httpd"
  Normal   Created    8s (x2 over 8m)     kubelet            Created container liveness
  Normal   Started    8s (x2 over 8m)     kubelet            Started container liveness
  Normal   Pulled     8s                  kubelet            Successfully pulled image "docker.io/httpd" in 828.473886ms (828.484388ms including wai
ting)
```

Kubectl get pods

```
controlplane $ kubectl get pods -w
NAME             READY   STATUS    RESTARTS      AGE
liveness-exec1   1/1     Running   1 (33s ago)   8m35s
```

The Number of restarts has been updated to "2", symbolising 2 restarts of the container.

READINESS PROBE

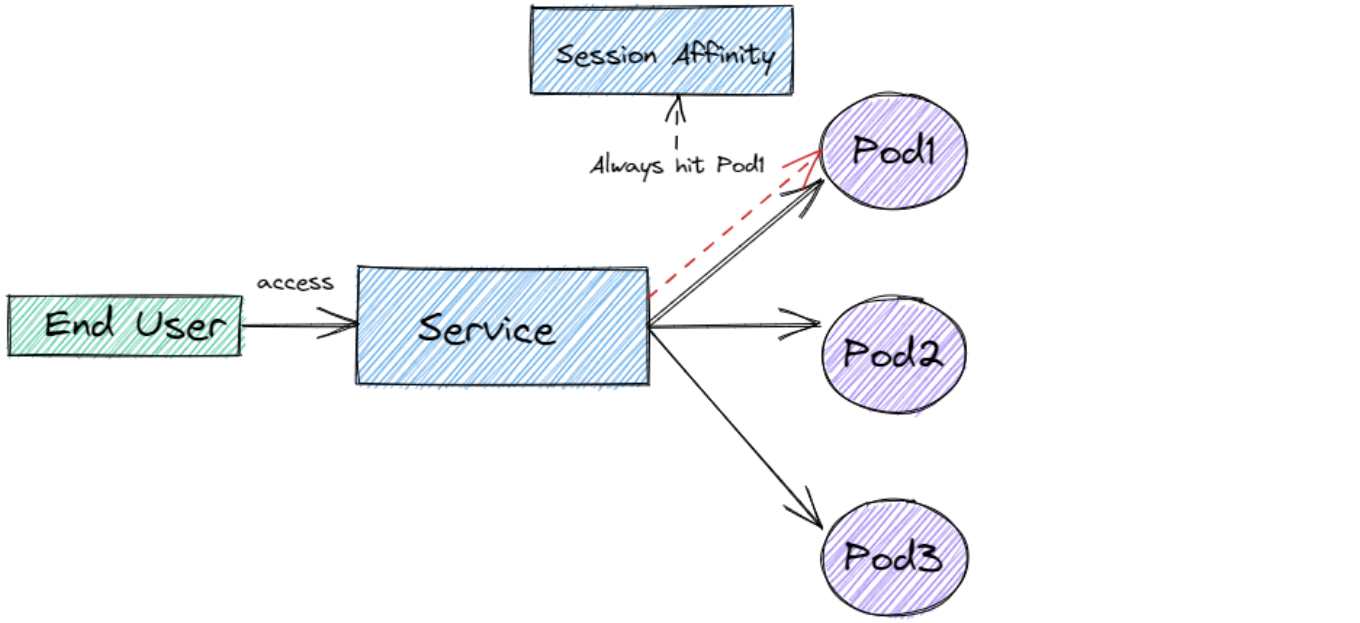
The kubelet uses readiness probes to know when a container is ready to start accepting traffic. A Pod is considered ready when all of its containers are ready. One use of this signal is to control which Pods are used as backends for Services. When a Pod is not ready, it is removed from Service load balancers.

1/4/23

SESSION AFFINITY

This makes sure that connections from a particular client are passed to the same Pod each time.
We can select the session affinity based on the client's IP addresses by setting **.spec.sessionAffinity to ClientIP for a Service** (the default is None).

Scenario:



When the end user will hit the service, it will take it to the pods in "Round Robin" Fashion by default.
But If the End User wants to access only the Pod1 everytime it hits the service, then we use SESSION AFFINITY.

Kube Proxy manages this thing, as it has all the details of the client's IP

Example:

Creating a Deployment "dep1" , with 3 replicas.
Creating a service of the deployment.

```
controlplane $ kubectl create deployment dep1 --image=docker.io/httpd --replicas=3
deployment.apps/dep1 created
controlplane $ kubectl get pods
NAME                    READY   STATUS    RESTARTS   AGE
dep1-859f6658bd-csjv6   1/1     Running   0          27s
dep1-859f6658bd-k2mhl   1/1     Running   0          27s
dep1-859f6658bd-tc2qs   1/1     Running   0          27s
controlplane $ kubectl expose deployment dep1 --port=80
service/dep1 exposed
controlplane $ kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
dep1         ClusterIP   10.110.149.62   <none>        80/TCP    17s
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP   28d
```