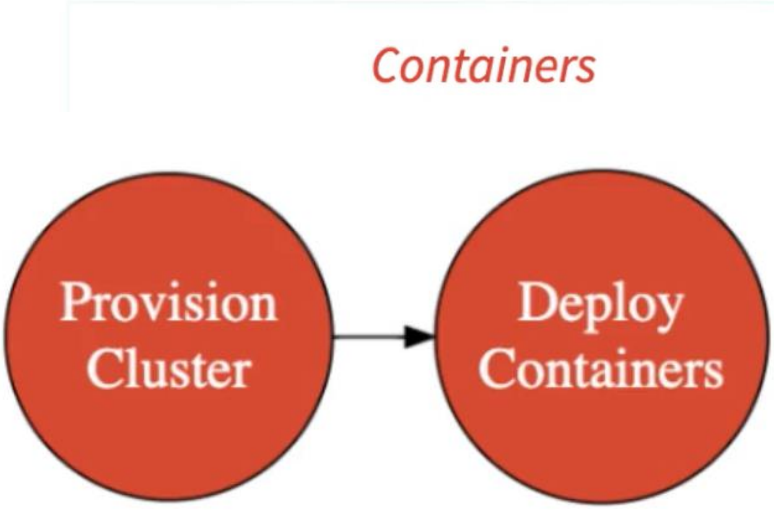
Containerization :
We have multiple microservices build on different languages .



Install docker .
Check docker version : docker –version

```
ubuntu@ip-172-31-38-131:~$ docker --version
Docker version 23.0.3, build 3e7cbfd
```

Run a python application in docker :

```
ubuntu@ip-172-31-38-131:~$ sudo docker run -p 5000:5000 in28min/hello-world-python:0.0.1.RELEASE
Unable to find image 'in28min/hello-world-python:0.0.1.RELEASE' locally
0.0.1.RELEASE: Pulling from in28min/hello-world-python
21c83c524219: Pull complete
9a80d14c35bd: Pull complete
0d32a27dde5a: Pull complete
2cb80a514e07: Pull complete
d5d3b19aaadd: Pull complete
694c09e178f0: Pull complete
2163a4c6fcc6: Pull complete
26893ad78bb3: Pull complete
Digest: sha256:a77f9165a81a3650f0211f823f7a5cdfdcb7b7e458cd193ea644ea10fb476fa2
Status: Downloaded newer image for in28min/hello-world-python:0.0.1.RELEASE
 * Serving Flask app 'launch' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses (0.0.0.0)
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000 (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 334-322-405
```

We are running above image from dockerhub .

To check the history of commands ran in docker image :

```
Error response from daemon: No such image: 81153c577577:latest
ubuntu@ip-172-31-38-131:~$ sudo docker images
REPOSITORY                          TAG             IMAGE ID       CREATED         SIZE
sravtar/cicd_docker_build_webapp    latest          6f87587c9cdf   5 days ago      301MB
sonarqube                           latest          ec2e9b64a080   8 days ago      685MB
in28min/hello-world-python          0.0.1.RELEASE   6d1dfe87a934   10 months ago   91MB
hello-world                         latest          feb5d9fea6a5   18 months ago   13.3kB
ubuntu@ip-172-31-38-131:~$ sudo docker history 6d1dfe87a934
IMAGE          CREATED         CREATED BY                                      SIZE      COMMENT
6d1dfe87a934   10 months ago   CMD ["/bin/sh" "-c" "python ./launch.py"]       0B        buildkit.dockerfile.v0
<missing>      10 months ago   EXPOSE map[5000/tcp:{}]                         0B        buildkit.dockerfile.v0
<missing>      10 months ago   RUN /bin/sh -c pip install -r requirements.t…   11MB      buildkit.dockerfile.v0
<missing>      10 months ago   COPY . /app # buildkit                          440B      buildkit.dockerfile.v0
<missing>      10 months ago   WORKDIR /app                                    0B        buildkit.dockerfile.v0
<missing>      2 years ago     /bin/sh -c #(nop)  CMD ["python3"]              0B
<missing>      2 years ago     /bin/sh -c set -ex;   wget -O get-pip.py "$P…   6.51MB
<missing>      2 years ago     /bin/sh -c #(nop)  ENV PYTHON_GET_PIP_SHA256…   0B
<missing>      2 years ago     /bin/sh -c #(nop)  ENV PYTHON_GET_PIP_URL=ht…   0B
<missing>      2 years ago     /bin/sh -c #(nop)  ENV PYTHON_PIP_VERSION=20…   0B
<missing>      2 years ago     /bin/sh -c cd /usr/local/bin  && ln -s idle3…   32B
<missing>      2 years ago     /bin/sh -c set -ex  && apk add --no-cache --…   67.4MB
<missing>      2 years ago     /bin/sh -c #(nop)  ENV PYTHON_VERSION=3.8.3     0B
<missing>      2 years ago     /bin/sh -c #(nop)  ENV GPG_KEY=E3FF2839C048B…   0B
<missing>      2 years ago     /bin/sh -c apk add --no-cache ca-certificates   551kB
<missing>      2 years ago     /bin/sh -c #(nop)  ENV LANG=C.UTF-8             0B
<missing>      2 years ago     /bin/sh -c #(nop)  ENV PATH=/usr/local/bin:/…   0B
<missing>      2 years ago     /bin/sh -c #(nop)  CMD ["/bin/sh"]              0B
<missing>      2 years ago     /bin/sh -c #(nop) ADD file:66a440394c2442570…   5.58MB
```

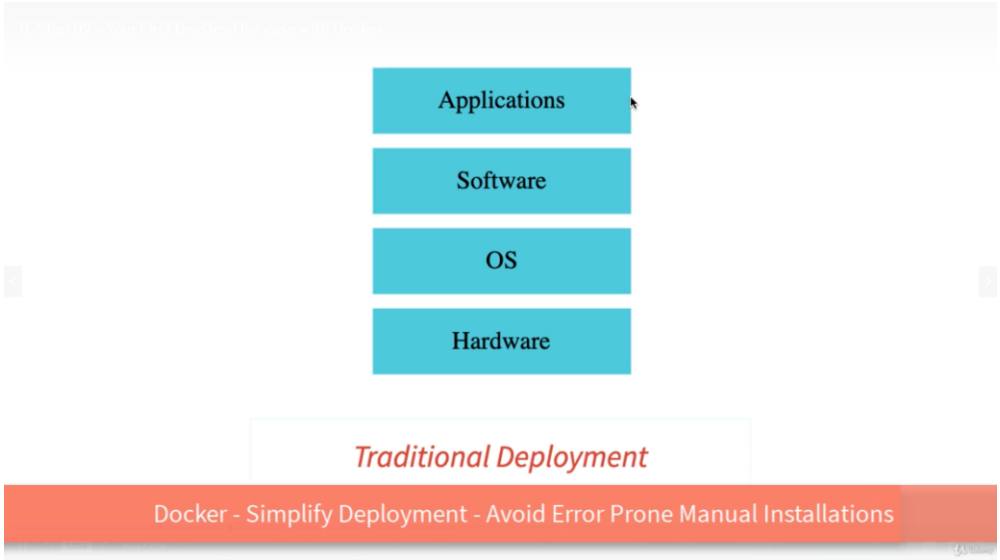
So the application is running on port 5000 using the image we specified .

```
156a4f961cd1   hello-world                                   "/hello"                 5 days ago      Exited (0) 5 days ago                focused_brahmagupta
ubuntu@ip-172-31-38-131:~$ sudo docker start 81153c577577
81153c577577
ubuntu@ip-172-31-38-131:~$ sudo docker ps -a
CONTAINER ID   IMAGE                                 COMMAND                  CREATED         STATUS                  PORTS                                          NAMES
81153c577577   in28min/hello-world-python:0.0.1.RELEASE   "/bin/sh -c 'python …"   7 minutes ago   Up 4 seconds            0.0.0.0:5000->5000/tcp, :::5000->5000/tcp      wizardly_northcutt
87d818210e42   sonarqube                             "/opt/sonarqube/dock…"   3 days ago      Exited (0) 3 days ago                                                  sonarqube
3eebb846443a   sravtar/cicd_docker_build_webapp      "/usr/sbin/httpd -D …"   5 days ago      Exited (0) 3 days ago                                                  scriptedcontainer
156a4f961cd1   hello-world                           "/hello"                 5 days ago      Exited (0) 5 days ago                                                  focused_brahmagupta
```

Not secure | 43.205.124.201:5000

```
// 20230412112721
// http://43.205.124.201:5000/

{
  "message": "Hello World Python v1"
}
```



With docker we don't need to worry about above components , there will be created in docker image .

Lets run java application in docker :

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

Once we run java image we get response from java application .

Not secure | 43.205.124.201:5000

```
// 20230412113643
// http://43.205.124.201:5000/

{
  "message": "Hello World Java v1"
}
```

Lets run nodejs application :

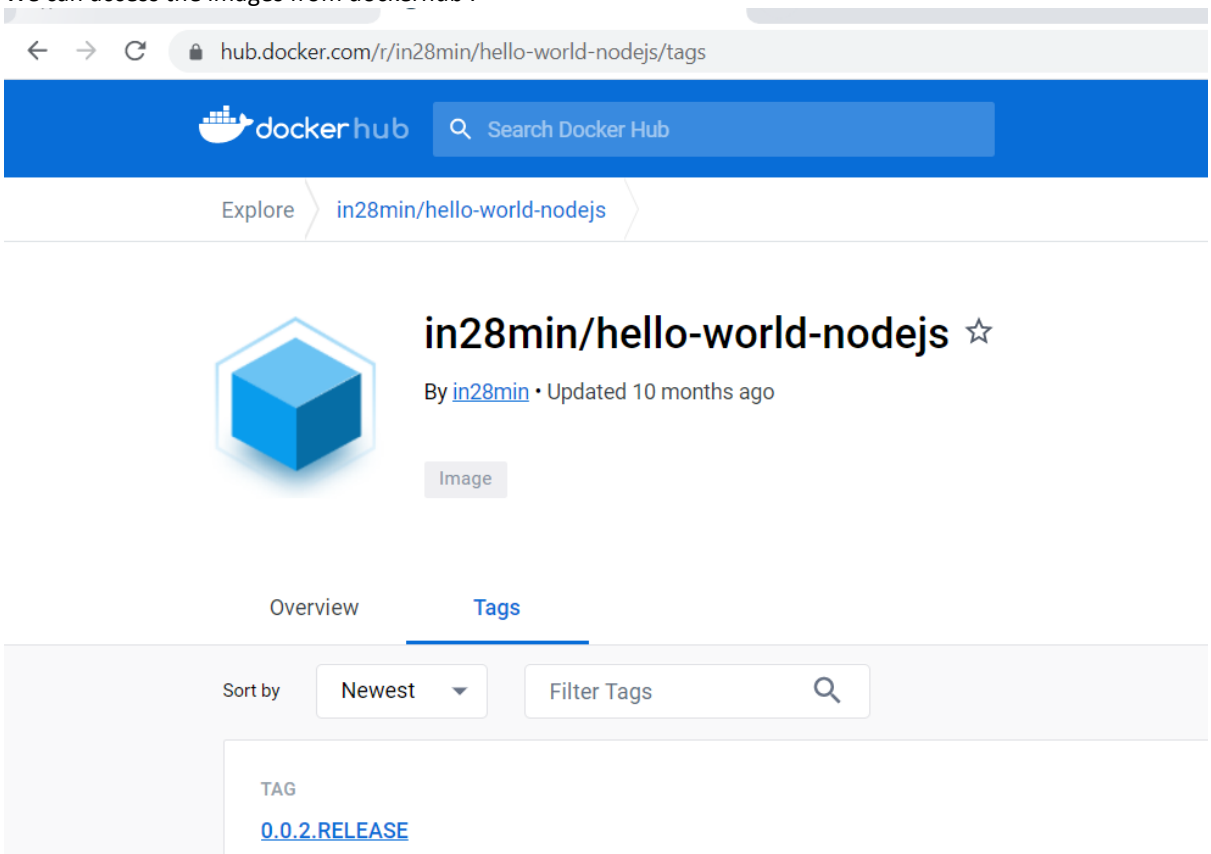
```
docker run -p 5000:5000 in28min/hello-world-nodejs:0.0.1.RELEASE
```

Not secure | 43.205.124.201:5000

```
// 20230412113954
// http://43.205.124.201:5000/

{
  "message": "Hello World JavaScript v1"
}
```

All the images we ran are stored in docker registry – dockerhub .
We can access the images from dockerhub .



Whenever we run a container it is a part of internal docker network – bridge network .
By default all container run inside bridge network .
We will not be able to access the container unless port is exposed .
At a same tym if we want to run different applications in docker then we have to map with different ports .

Running a container in detached mode .

```
upta
ubuntu@ip-172-31-38-131:~$ sudo docker stop b8032a5d8442
b8032a5d8442
ubuntu@ip-172-31-38-131:~$ sudo docker rm b8032a5d8442
b8032a5d8442
ubuntu@ip-172-31-38-131:~$ sudo docker run -d -p 5000:5000 in28min/hello-world-nodejs:0.0.1.RELEASE
fcff9a5b616246a589b03388d47b19d2b941a2ae50d1471e187a903c2df5415d
ubuntu@ip-172-31-38-131:~$
```

When we run container in detached mode we will not be able to see logs of container .
We can check logs of container .

```
ubuntu@ip-172-31-38-131:~$ sudo docker logs fcff9a5b6162
Ready on port 5000!
ubuntu@ip-172-31-38-131:~$
```

Terminal attached with logs :

```
ubuntu@ip-172-31-38-131:~$ sudo docker logs -f fcff9a5b6162
Ready on port 5000!
```

Get list of images present :

```
ubuntu@ip-172-31-38-131:~$ sudo docker images
REPOSITORY                          TAG             IMAGE ID       CREATED          SIZE
sravtar/cicd_docker_build_webapp    latest          6f87587c9cdf   5 days ago       301MB
sonarqube                           latest          ec2e9b64a080   8 days ago       685MB
in28min/hello-world-java            0.0.1.RELEASE   4f6bc0e79b5b   10 months ago    122MB
in28min/hello-world-nodejs          0.0.1.RELEASE   3ea2933d6387   10 months ago    104MB
in28min/hello-world-python          0.0.1.RELEASE   6d1dfe87a934   10 months ago    91MB
hello-world                         latest          feb5d9fea6a5   18 months ago    13.3kB
ubuntu@ip-172-31-38-131:~$
```

Get list of container running :

```
ubuntu@ip-172-31-38-131:~$ sudo docker container ls
CONTAINER ID   IMAGE                                  COMMAND                  CREATED         STATUS         PORTS                                       NAMES
fcff9a5b6162   in28min/hello-world-nodejs:0.0.1.RELEASE   "docker-entrypoint.s…"   8 minutes ago   Up 8 minutes   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   sharp_ramanujan
ubuntu@ip-172-31-38-131:~$
```

Get list of all containers :

```
ubuntu@ip-172-31-38-131:~$ sudo docker container ls -a
CONTAINER ID   IMAGE                                  COMMAND                  CREATED          STATUS                      PORTS                                       NAMES
fcff9a5b6162   in28min/hello-world-nodejs:0.0.1.RELEASE   "docker-entrypoint.s…"   10 minutes ago   Up 9 minutes                0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   sharp_ramanu
jan
031e80410983   in28min/hello-world-java:0.0.1.RELEASE     "sh -c 'java -jar /h…"   52 minutes ago   Exited (130) 46 minutes ago                                             quizzical_ga
uss
81153c577577   in28min/hello-world-python:0.0.1.RELEASE   "/bin/sh -c 'python …"   About an hour ago   Exited (0) 53 minutes ago                                               wizardly_nor
thcutt
87d818210e42   sonarqube                                  "/opt/sonarqube/dock…"   3 days ago       Exited (0) 3 days ago                                                   sonarqube
3eebb846443a   sravtar/cicd_docker_build_webapp           "/usr/sbin/httpd -D …"   5 days ago       Exited (0) 3 days ago                                                   scriptedcont
ainer
156a4f961cd1   hello-world                                "/hello"                 5 days ago       Exited (0) 5 days ago                                                   focused_brah
magupta
ubuntu@ip-172-31-38-131:~$
```

Docker architecture :

Docker client – docker daemon (server component). – container , local images , image registry .

Whenever we run any command , docker client sends it to docker daemon , docker doeamon is responsible for execution of specific command .
Docker daemon is responsible for – managing containers , managind images , pushing images to image repository .
So if we run an container – docker client send request to docker doeamon , daemon check if image is present locally or not , If its present locally then it will not go to docker registry .

Docker :

1) Standardized application packaging
2) Multi platform support .
3) Light weight and isolation .

Instead of docker run we can also use docker pull . : docker pull mysql – it will pull latest tag image .

```
ubuntu@ip-172-31-38-131:~$ sudo docker pull mysql
Using default tag: latest
latest: Pulling from library/mysql
328ba678bf27: Pull complete
f3f5ff008d73: Pull complete
dd7054d6d0c7: Pull complete
70b5d4e8750e: Pull complete
cdc4a7b43bdd: Pull complete
3e9c0b61a8f3: Pull complete
806a08b6c085: Pull complete
021b2cebd832: Pull complete
ad31ba45b26b: Pull complete
0d4c2bd59d1c: Pull complete
148dcef42e3b: Pull complete
Digest: sha256:f496c25da703053a6e0717f1d52092205775304ea57535cc9fcaa6f35867800b
Status: Downloaded newer image for mysql:latest
docker.io/library/mysql:latest
ubuntu@ip-172-31-38-131:~$
```

Pull command only pull the image , it will not run docker container .

We can also search for all images present .

```
ubuntu@ip-172-31-38-131:~$ sudo docker search mysql
NAME                            DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
mysql                           MySQL is a widely used, open-source relation…   14029     [OK]
mariadb                         MariaDB Server is a high performing open sou…   5352      [OK]
percona                         Percona Server is a fork of the MySQL relati…   603       [OK]
phpmyadmin                      phpMyAdmin - A web interface for MySQL and M…   780       [OK]
circleci/mysql                  MySQL is a widely used, open-source relation…   29
bitnami/mysql                   Bitnami MySQL Docker Image                      82                   [OK]
bitnami/mysqld-exporter                                                         4
ubuntu/mysql                    MySQL open source fast, stable, multi-thread…   45
cimg/mysql                                                                      0
rapidfort/mysql                 RapidFort optimized, hardened image for MySQL   14
google/mysql                    MySQL server for Google Compute Engine          23                   [OK]
rapidfort/mysql8-ib             RapidFort optimized, hardened image for MySQ…   0
hashicorp/mysql-portworx-demo                                                   0
rapidfort/mysql-official        RapidFort optimized, hardened image for MySQ…   0
newrelic/mysql-plugin           New Relic Plugin for monitoring MySQL databa…   1                    [OK]
databack/mysql-backup           Back up mysql databases to... anywhere!         82
linuxserver/mysql               A Mysql container, brought to you by LinuxSe…   38
bitnamicharts/mysql                                                             0
mirantis/mysql                                                                  0
docksal/mysql                   MySQL service images for Docksal - https://d…   0
vitess/mysqlctld                vitess/mysqlctld                                1                    [OK]
linuxserver/mysql-workbench                                                     48
eclipse/mysql                   Mysql 5.7, curl, rsync                          0                    [OK]
drud/mysql                                                                      0
ilios/mysql                     Mysql configured for running Ilios              1                    [OK]
ubuntu@ip-172-31-38-131:~$
```

It will give list of all mysql images from registry .

We can get more information about image using docker inspect :

```
ubuntu@ip-172-31-38-131:~$ sudo docker inspect 412b8cc72e4a
[
    {
        "Id": "sha256:412b8cc72e4a28e086097c3fcb1ca391beaefe86bc421a57bc53f7596461ce3b",
        "RepoTags": [
            "mysql:latest"
        ],
        "RepoDigests": [
            "mysql@sha256:f496c25da703053a6e0717f1d52092205775304ea57535cc9fcaa6f35867800b"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2023-04-06T18:45:06.484463927Z",
        "Container": "2b2da17aa1c6bd4b1ccc0ac312c2ab5e9c3ccf35de3b630848710e89932310fd",
        "ContainerConfig": {
            "Hostname": "2b2da17aa1c6",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "ExposedPorts": {
                "3306/tcp": {},
                "33060/tcp": {}
            },
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
                "GOSU_VERSION=1.16",
                "MYSQL_MAJOR=8.0",
                "MYSQL_VERSION=8.0.32-1.el8",
```

To remove image :

```
ubuntu@ip-172-31-38-131:~$ sudo docker image remove mysql
Untagged: mysql:latest
Untagged: mysql@sha256:f496c25da703053a6e0717f1d52092205775304ea57535cc9fcaa6f35867800b
Deleted: sha256:412b8cc72e4a28e086097c3fcb1ca391beaefe86bc421a57bc53f7596461ce3b
Deleted: sha256:699b29bcc4724e503bc81a1ec5a651577acf4f705ea229373ff04fa1b8e5e928
Deleted: sha256:9dfec1ef6d43e55f98851f4225c550bacff9e3a65aee6cbb28aa471e49642039
Deleted: sha256:c307d60bfcdbd90f424b4072638115de08a5a18579e53276639fc4085caa3394
Deleted: sha256:f08b9c9816cd2c2804d06abb350d0c086762471ecda5358ef4198e313f856749
Deleted: sha256:942885f4a39dc56ff998368098c25901f4dbd351122cdb5d889d584865d607a6
Deleted: sha256:ddecc532f6753873e59e3cc720e22196934bb0d08072a113a876b0577afd14c0
Deleted: sha256:d023b92a46a5fa8fa8d54387e6d3cb0c73997fefc64ec9000eab0ee1c550ef45
Deleted: sha256:f1c1643119168a94089eab1c9126cda0ee6056a4bb4b18e27a7dcacdf4823972
Deleted: sha256:b147319dd21e8994e6d2fb3bb58a8278c5a72f39488e1f1cff94fc73f1089eb9
Deleted: sha256:ff7c2b28c0dfaa63d0d30b7a5069bf526b0f6492143110381351bbf7d07b4baf
Deleted: sha256:caefa4e45110eab274ebbdbc781f9227229f947f8718cee62ebeff1aac8f1d5b
ubuntu@ip-172-31-38-131:~$
```

Remove container :

```
upta
ubuntu@ip-172-31-38-131:~$ sudo docker container rm 031e80410983
031e80410983
ubuntu@ip-172-31-38-131:~$
```

List all container : docker container ls
Pause container : docker container pause container_id : it will not server any request .
Unpause : docker container unpause container_id : it will server request
Kill : docker container kill container_id – immediately kill application .
Inspect container : docker container inspect container_id

TO remove all stopped containers : docker container prune
Disc usage of docker : docker system df
Get real time event from server : docker system events – all events happening in containers .
To delete all stopped container , images which are not associated with any container : docker system prune -a

docker stats container_id : it give stats about specific container .
To limit container to use defined memory only : docker container run -p 5000:5000 -d -m 512m image:tag

Limit amount of cpu container uses :
First stop container
docker container run -p 5000:5000 -d -m 512m --cpu-quota=50000 image:tag

total cpu quota is 100000

Build docker image for python application using dockerfile :

```
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-python$ sudo docker build -t sravtar/hello-world-python:0.0.2.RELEASE .
[+] Building 8.4s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 243B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/python:alpine3.10
 => [1/4] FROM docker.io/library/python:alpine3.10@sha256:152b1952d4b42e360f2efd3037df9b645328c0cc6fbe9c63decbffbff407b96a
 => => resolve docker.io/library/python:alpine3.10@sha256:152b1952d4b42e360f2efd3037df9b645328c0cc6fbe9c63decbffbff407b96a
 => => sha256:152b1952d4b42e360f2efd3037df9b645328c0cc6fbe9c63decbffbff407b96a 1.65kB / 1.65kB
 => => sha256:655edcda221823fcdb79b61095dd77e6c767bf1543505dcf078f6945497c7fcf 1.37kB / 1.37kB
 => => sha256:cf02325838736bb42f50cff8931c1b0e5363d6106283d98ba769ac81376e9125 7.14kB / 7.14kB
 => [internal] load build context
 => => transferring context: 565B
 => [2/4] WORKDIR /app
 => [3/4] COPY . /app
 => [4/4] RUN pip install -r requirements.txt
 => exporting to image
 => => exporting layers
 => => writing image sha256:71cc626604ad19038855687ab9cb1fb3e96e90f7778bc97a9a62e8bf262634f6
 => => naming to docker.io/sravtar/hello-world-python:0.0.2.RELEASE
```

Image is build .

Lets run container using image .

```
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-python$ sudo docker run -d -p 5000:5000 sravtar/hello-world-python:0.0.2.RELEASE
b2075a3d177f8f691a8691fe3f84ba777bcae2ef43b7ef772fef38ffcad1c315
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-python$
```

Not secure | 3.110.195.97:5000

```
// 20230412163016
// http://3.110.195.97:5000/

{
  "message": "Hello World Python v2"
}
```

Application is accessible .

Push the image to dockerhub :

```
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-python$ sudo 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://hub.docker.com to create one.
Username: sravtar
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@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-python$ sudo docker push sravtar/hello-world-python:0.0.2.RELEASE
The push refers to repository [docker.io/sravtar/hello-world-python]
e229adfa3bcd: Pushed
8ea1cf05200a: Pushed
8355c3efb132: Pushed
4e633e2489a3: Mounted from library/python
798f2bf6d71c: Mounted from library/python
e1c1f46b85cc: Mounted from library/python
057be770731c: Mounted from library/python
1b3ee35aacca: Mounted from library/python
0.0.2.RELEASE: digest: sha256:d4414e080e1de1a6e768911416b6ad6a7021af231ff13a19efb0e77f115d9596 size: 1992
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-python$
```

sravtar / hello-world-python

Description

This repository does not have a description

Last pushed: a minute ago

Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
0.0.2.RELEASE		Image	---	a minute ago

Image pushed to dockerhub .

Build and push dockerimage for nodejs application .
In index.js – code is specified .
In package.json – dependencies are mentioned .

In dockerfile we specify how image to be build , we can use pip , maven , gradle to build image .

```
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-nodejs$ sudo docker build -t sravtar/hello-world-nodejs:0.0.2.RELEASE .
[+] Building 17.5s (10/10) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 192B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/node:8.16.1-alpine
 => [auth] library/node:pull token for registry-1.docker.io
 => [1/4] FROM docker.io/library/node:8.16.1-alpine@sha256:e1d58a32a7303b3f95f64fe13f2c6679e42879f02a2b77e06771a023e7706e02
 => => resolve docker.io/library/node:8.16.1-alpine@sha256:e1d58a32a7303b3f95f64fe13f2c6679e42879f02a2b77e06771a023e7706e02
 => => sha256:e1d58a32a7303b3f95f64fe13f2c6679e42879f02a2b77e06771a023e7706e02 1.65kB / 1.65kB
 => => sha256:4201be67ca08fbd095790ea514766ec96ef1debd73b9858922d47862ccd242e2 1.16kB / 1.16kB
 => => sha256:b9e61ad789af5331d65e2bdb7493f5240ec9a298b78511d4da6697500c931c34 5.66kB / 5.66kB
 => [internal] load build context
 => => transferring context: 992B
 => [2/4] WORKDIR /app
 => [3/4] COPY . /app
 => [4/4] RUN npm install
 => exporting to image
 => => exporting layers
 => => writing image sha256:bc432f6b454ecf91997f22a262ec3bdffe79cfc99cd0d613f79ac111748ef55b
 => => naming to docker.io/sravtar/hello-world-nodejs:0.0.2.RELEASE
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-nodejs$ sudo docker container run -d -p 5000:5000 sravtar/hello-world-nodejs:0.0.2.RELEASE
9f86c53693afaa8d1bb2ea1895f40795dc21f7d96e5b1b0384c782ba2314220f
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-nodejs$
```

So image is build and container run .

Not secure | 3.110.195.97:5000

```
// 20230412165324
// http://3.110.195.97:5000/

{
  "message": "Hello World JavaScript v1"
}
```

Application is accessible .

Push image to docker hub .

```
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-nodejs$ sudo docker push sravtar/hello-world-nodejs:0.0.2.RELEASE
The push refers to repository [docker.io/sravtar/hello-world-nodejs]
be48ec6943e4: Pushed
f5805c279750: Pushed
a66f01355b7b: Pushed
e7ae04d3f37c: Mounted from library/node
e29ab5067804: Mounted from library/node
ae4ceb8dc557: Mounted from library/node
f1b5933fe4b5: Mounted from library/node
0.0.2.RELEASE: digest: sha256:bc0d38e5f947ead99eb0eb95d29efd466a7f8e0a80bad176b4e2cde80cc4e7c8 size: 1783
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-nodejs$
```

Build and push docker image for java application :

For java application we are using 2 stage dockerfile :

Pom.xml contains all dependencies .

```
Dockerfile  pom-xml-document  pom.xml  src
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-java$ cat Dockerfile
# Build a JAR File
FROM maven:3.8.2-jdk-8-slim AS stage1
WORKDIR /home/app
COPY . /home/app/
RUN mvn -f /home/app/pom.xml clean package

# Create an Image
FROM openjdk:8-jdk-alpine
EXPOSE 5000
COPY --from=stage1 /home/app/target/hello-world-java.jar hello-world-java.jar
ENTRYPOINT ["sh", "-c", "java -jar /hello-world-java.jar"]

#This Dockerfile creates a Docker image for a Java application. The application code is built using Maven in the first stage, and the resulting JAR file is then copied to the second stage w
hich runs a minimal Alpine-based OpenJDK 8 container.
#Here is a breakdown of the steps in the Dockerfile:

#FROM maven:3.8.2-jdk-8-slim AS stage1: This starts the first stage of the multi-stage build process, using a slim version of Maven 3.8.2 with OpenJDK 8 as the base image.

#WORKDIR /home/app: Sets the working directory to /home/app in the container.

#COPY . /home/app/: Copies the contents of the current directory (presumably the source code for the Java application) to /home/app in the container.

#RUN mvn -f /home/app/pom.xml clean package: Runs the Maven clean and package goals on the pom.xml file located in /home/app directory, producing a JAR file with the compiled Java code in t
he /home/app/target directory.

#FROM openjdk:8-jdk-alpine: This starts the second stage of the build process, using a minimal Alpine-based OpenJDK 8 image as the base image.

#EXPOSE 5000: Exposes port 5000 on the container.

#COPY --from=stage1 /home/app/target/hello-world-java.jar hello-world-java.jar: Copies the JAR file produced in the first stage from /home/app/target directory to the root directory in the
second stage.

#ENTRYPOINT ["sh", "-c", "java -jar /hello-world-java.jar"]: Sets the command to be run when the container starts. This command starts a shell and runs the Java program with the JAR file lo
cated at /hello-world-java.jar.
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-java$
```

To deploy java application we first create jar file and then we run jar file in our image .

In dockerfile : 1st stage – to build jar using maven ; 2nd stage : copy jar file to image .

And image will be used to run java application .
We link two stages by making output of 1 stage and using it as input in second stage .
We are using maven base image .

Build the image : docker build -t imagename:version .

```
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-java$ sudo docker build -t sravtar/hello-world-java:0.0.2.RELEASE .
[+] Building 1.5s (14/14) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 1.91kB
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/openjdk:8-jdk-alpine
 => [internal] load metadata for docker.io/library/maven:3.8.2-jdk-8-slim
 => [auth] library/openjdk:pull token for registry-1.docker.io
 => [auth] library/maven:pull token for registry-1.docker.io
 => [stage1 1/4] FROM docker.io/library/maven:3.8.2-jdk-8-slim@sha256:1a789af298566fe11c6b1dec7f48bc968ade563b13769952aeaf5ad77d2665dd
 => [stage-1 1/2] FROM docker.io/library/openjdk:8-jdk-alpine@sha256:94792824df2df33402f201713f932b58cb9de94a0cd524164a0f2283343547b3
 => [internal] load build context
 => => transferring context: 1.28kB
 => CACHED [stage1 2/4] WORKDIR /home/app
 => CACHED [stage1 3/4] COPY . /home/app/
 => CACHED [stage1 4/4] RUN mvn -f /home/app/pom.xml clean package
 => CACHED [stage-1 2/2] COPY --from=stage1 /home/app/target/hello-world-java.jar hello-world-java.jar
 => exporting to image
 => => exporting layers
 => => writing image sha256:b619d706370a2bb6b41ad3d45824f029e7553b3221d06d4321e80c223ec70990
 => => naming to docker.io/sravtar/hello-world-java:0.0.2.RELEASE
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-java$
```

Lets run image as container : docker run -d -p 5000:5000 imagename:version

```
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-java$ sudo docker run -d -p 5000:5000 sravtar/hello-world-java:0.0.2.RELEASE
7ebd99a8a6f9fc3ff0b0b73afc47886cd9a29d16a8a5bb5f5e270cd61d20d8b3
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-java$
```

Push docker image : docker push imagename

```
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-java$ sudo docker push sravtar/hello-world-java:0.0.2.RELEASE
The push refers to repository [docker.io/sravtar/hello-world-java]
24cf3098d992: Pushed
ceaf9e1ebef5: Pushed
9b9b7f3d56a0: Mounted from in28min/hello-world-java
f1b5933fe4b5: Mounted from in28min/hello-world-nodejs
0.0.2.RELEASE: digest: sha256:a363d8d201b0e8a32446b8801c1cb580094c970b0b074f6de1237414c7ab0b0e size: 1159
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-java$
```

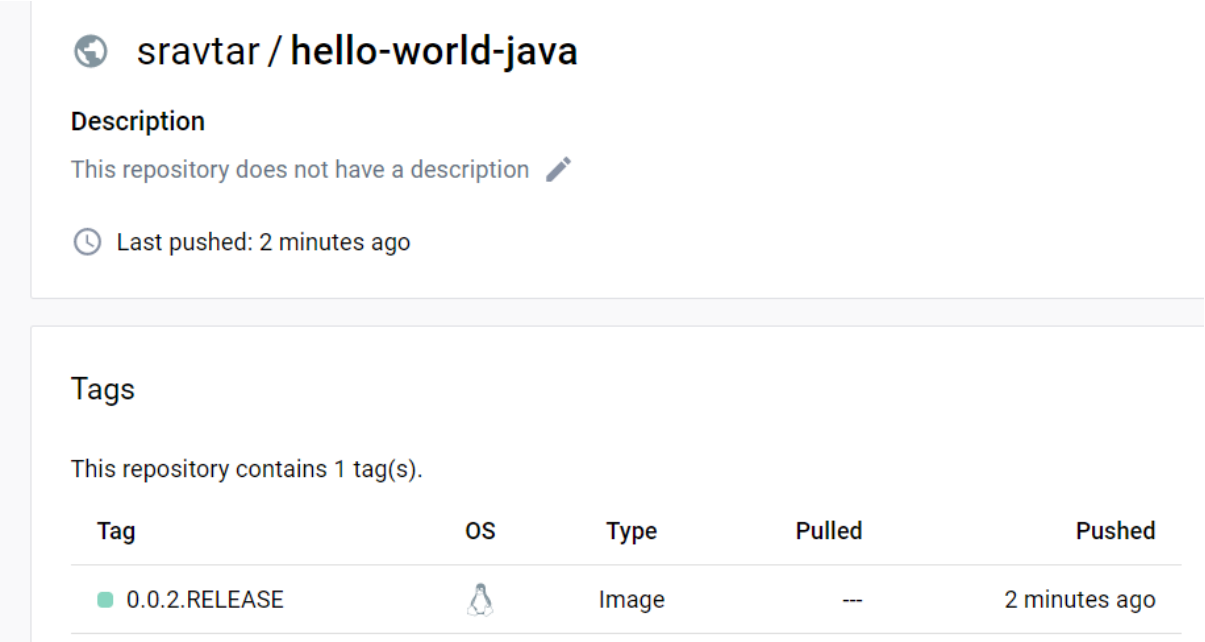


Image pushed .

Build efficient docker images : improve layer caching
If we build same image again then everything will be build from cache .

```
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-nodejs$ sudo docker build -t sravtar/hello-world-nodejs .
[+] Building 0.7s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 192B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/node:8.16.1-alpine
 => [1/4] FROM docker.io/library/node:8.16.1-alpine@sha256:e1d58a32a7303b3f95f64fe13f2c6679e42879f02a2b77e06771a023e7706e02
 => [internal] load build context
 => => transferring context: 92B
 => CACHED [2/4] WORKDIR /app
 => CACHED [3/4] COPY . /app
 => CACHED [4/4] RUN npm install
 => exporting to image
 => => exporting layers
 => => writing image sha256:79c9ff10c403f985da4dabe3b7e91dff4133d3085391df0b168e6245f164c3e2
 => => naming to docker.io/sravtar/hello-world-nodejs
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-nodejs$
```

It will pick data from cache until any change in dockerfile is not encountered .
Lets change code anf build image again .

```
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-nodejs$ sudo docker build -t sravtar/hello-world-nodejs .
[+] Building 15.0s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 192B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/node:8.16.1-alpine
 => [1/4] FROM docker.io/library/node:8.16.1-alpine@sha256:e1d58a32a7303b3f95f64fe13f2c6679e42879f02a2b77e06771a023e7706e02
 => [internal] load build context
 => => transferring context: 463B
 => CACHED [2/4] WORKDIR /app
 => [3/4] COPY . /app
 => [4/4] RUN npm install
 => exporting to image
 => => exporting layers
 => => writing image sha256:b12d09a570d9bc2b0939f9eb70343b13c202b6cf40bec6fb0de8c4577833e0c7
 => => naming to docker.io/sravtar/hello-world-nodejs
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-nodejs$
```

So even if code is changed it will build dependency layer also again since dependency is build after code layer .

Developers usually change code but we don't change dependency frequently , so how do we create dependency as separate layer ?
So we will first copy dependeny file only and will run dependency after that we will build whole code .

```
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-nodejs$ cat Dockerfile
FROM node:8.16.1-alpine
WORKDIR /app
COPY package.json /app
RUN npm install
EXPOSE 5000
COPY . /app
CMD node index.js

#ENTRYPOINT ["node", "index.js"]
#COPY package.json /app
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-nodejs$
```

In above dockerfile we first build dependency as separate layer and after that we will copy whole code , as code may change frequently .

```
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-nodejs$ sudo docker build -t sravtar/hello-world-nodejs .
[+] Building 1.5s (11/11) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 215B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/node:8.16.1-alpine
 => [auth] library/node:pull token for registry-1.docker.io
 => [1/5] FROM docker.io/library/node:8.16.1-alpine@sha256:e1d58a32a7303b3f95f64fe13f2c6679e42879f02a2b77e06771a023e7706e02
 => [internal] load build context
 => => transferring context: 463B
 => CACHED [2/5] WORKDIR /app
 => CACHED [3/5] COPY package.json /app
 => CACHED [4/5] RUN npm install
 => [5/5] COPY . /app
 => exporting to image
 => => exporting layers
 => => writing image sha256:337ce0185638abd136556967e4b1434061184d10668b58ba0af9f5e9549cc625
 => => naming to docker.io/sravtar/hello-world-nodejs
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-nodejs$
```

As we can see dependencies are build using cache and code is build separately .

```
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-nodejs$ sudo docker run -d -p 5000:5000 sravtar/hello-world-nodejs
562541812fd29420ac9ea81a065840a075547dd41c9b49a668a440f4bafe176d
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-nodejs$
```

Run the image .

Not secure | 13.233.186.206:5000

```
// 20230413121154
// http://13.233.186.206:5000/

{
  "message": "Hello World JavaScript v3"
}
```

V3 is deployed .

Similarly lets improve layering for python image also .

```
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-python$ cat Dockerfile
FROM python:alpine3.10
WORKDIR /app
COPY requirements.txt /app/requirements.txt
RUN pip install -r requirements.txt
EXPOSE 5000
COPY . /app
CMD python ./launch.py

#COPY requirements.txt /app/requirements.txt
#ENTRYPOINT ["python", "./launch.py"]
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-python$
```

Build requremnet.txt before whole code .

```
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-python$ sudo docker build -t sravtar/hello-world-python:0.0.2.RELEASE .
[+] Building 0.8s (10/10) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 287B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/python:alpine3.10
 => [1/5] FROM docker.io/library/python:alpine3.10@sha256:152b1952d4b42e360f2efd3037df9b645328c0cc6fbe9c63decbffbff407b96a
 => [internal] load build context
 => => transferring context: 339B
 => CACHED [2/5] WORKDIR /app
 => CACHED [3/5] COPY requirements.txt /app/requirements.txt
 => CACHED [4/5] RUN pip install -r requirements.txt
 => [5/5] COPY . /app
 => exporting to image
 => => exporting layers
 => => writing image sha256:e4ff6f5639d8301aa730f5b9a5b6e532ca79ea057106af00ead29bb74a71efd3
 => => naming to docker.io/sravtar/hello-world-python:0.0.2.RELEASE
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/hello-world/hello-world-python$
```

As we can see all layers are build from cache and new code is copied .

So always make sure that layers are cached as much as possible because it reduces the time for build , push , pull .

Entrypoint vs CMD :

To launch java application we used entrypoint and for others we used cmd .

Lets run : sudo docker run -d -p 5000:5000 sravtar/hello-world-nodejs ping google.com

It will run ping.google.com instead of command specified in DOCKERFILE in CMD .
Application wont run. Since we have specified CMD for nodejs dockerfile .

Lets run java application by specifying ping.google.com : sudo docker run -d -p 5000:5000 sravtar/hello-world-java:0.0.2.RELEASE ping google.com

Not secure | 13.233.186.206:5000

```
// 20230413123953
// http://13.233.186.206:5000/

{
  "message": "Hello World Java v1"
}
```

Now we can see that application is accessible and ping command is not overridden since we have specified entrypoint instead od CMD in dockerfile .

So, what is the difference between CMD and ENTRYPOINT?
The thing is, with CMD, whatever you pass from the command line will replace the instructions you wanted to execute.
What happens with CMD is, these parameters will be replacedby whatever you are passing in here.
So the command, the application will not be launched up and that will be replaced by ping google.com.
However, ENTRYPOINT does not worry about command line arguments.
Now, when to use an ENTRYPOINT and when to use a command? Let's say there is a chance that, instead of running node index.js, there might be a new file, which might be used to run the entire application. So, let's say index1.js, index2.js, index3.js. In that kind of situation, you can use CMD, so that you can override the command which you would want to execute in here. But, when you don't want to override the command, when you want this to be static, when you want every time, hello-world-java.jar is the first thing that should be executed when this application is launched up, then you'd go for an ENTRYPOINT.

DOCKER AND MICROSERVICES :
Docker helps in solving challenges related to microservices .

Instead of 1 large monolith we will build many small microservices .
We will use 2 microservices for our projects .

Running microservices as docker container :
Build image for both microservice :

```
icroservice-basic$ sudo docker build -t sravtar/currency-exchange:0.0.1-RELEASE .
[+] Building 82.6s (11/13)
 => [internal] load build definition from Dockerfile                                                                    0.0s
 => => transferring dockerfile: 352B                                                                                    0.0s
 => [internal] load .dockerignore                                                                                       0.1s
 => => transferring context: 2B                                                                                         0.0s
 => [internal] load metadata for docker.io/library/openjdk:8-jdk-alpine                                                 1.5s
 => [internal] load metadata for docker.io/library/maven:3.8.2-jdk-8-slim                                               1.5s
 => [auth] library/openjdk:pull token for registry-1.docker.io                                                          0.0s
 => [auth] library/maven:pull token for registry-1.docker.io                                                            0.0s
 => [build 1/4] FROM docker.io/library/maven:3.8.2-jdk-8-slim@sha256:1a789af298566fe11c6b1dec7f48bc968ade563b13769952aeaf5ad77d2665dd  0.0s
 => CACHED [stage-1 1/2] FROM docker.io/library/openjdk:8-jdk-alpine@sha256:94792824df2df33402f201713f932b58cb9de94a0cd524164a0f2283343547b3  0.0s
 => CACHED [build 2/4] WORKDIR /home/app                                                                                0.0s
 => [internal] load build context                                                                                       0.0s
 => => transferring context: 36.41kB                                                                                    0.0s
 => [build 3/4] COPY . /home/app                                                                                        0.1s
 => [build 4/4] RUN mvn -f /home/app/pom.xml clean package                                                             80.8s
 => => # Downloading from central: https://repo.maven.apache.org/maven2/org/eclipse/jetty/jetty-bom/9.4.19.v20190610/jetty-bom-9.4.19.v20190610.pom
 => => # Downloaded from central: https://repo.maven.apache.org/maven2/org/eclipse/jetty/jetty-bom/9.4.19.v20190610/jetty-bom-9.4.19.v20190610.pom (18 kB at 1.6 MB/s)
 => => # Downloading from spring-snapshots: https://repo.spring.io/snapshot/org/springframework/spring-framework-bom/5.1.9.RELEASE/spring-framework-bom-5.1.9.RELEASE.pom
 => => # Downloading from spring-milestones: https://repo.spring.io/milestone/org/springframework/spring-framework-bom/5.1.9.RELEASE/spring-framework-bom-5.1.9.RELEASE.pom
 => => # Downloading from central: https://repo.maven.apache.org/maven2/org/springframework/spring-framework-bom/5.1.9.RELEASE/spring-framework-bom-5.1.9.RELEASE.pom
 => => # Progress (1): 4.1/5.3 kB
```

```
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/microservices/02-currency-conversion-microservice-basic$ sudo docker build -t sravtar/currency-conversion:0.0.1-RELEASE .
[+] Building 280.4s (14/14) FINISHED
 => [internal] load build definition from Dockerfile                                                                    0.1s
 => => transferring dockerfile: 352B                                                                                    0.0s
 => [internal] load .dockerignore                                                                                       0.0s
 => => transferring context: 2B                                                                                         0.0s
 => [internal] load metadata for docker.io/library/openjdk:8-jdk-alpine                                                 1.5s
 => [internal] load metadata for docker.io/library/maven:3.8.2-jdk-8-slim                                               1.4s
 => [auth] library/maven:pull token for registry-1.docker.io                                                            0.0s
 => [auth] library/openjdk:pull token for registry-1.docker.io                                                          0.0s
 => [build 1/4] FROM docker.io/library/maven:3.8.2-jdk-8-slim@sha256:1a789af298566fe11c6b1dec7f48bc968ade563b13769952aeaf5ad77d2665dd  0.0s
 => CACHED [stage-1 1/2] FROM docker.io/library/openjdk:8-jdk-alpine@sha256:94792824df2df33402f201713f932b58cb9de94a0cd524164a0f2283343547b3  0.0s
 => [internal] load build context                                                                                       0.0s
 => => transferring context: 19.59kB                                                                                    0.0s
 => CACHED [build 2/4] WORKDIR /home/app                                                                                0.0s
 => [build 3/4] COPY . /home/app                                                                                        0.1s
 => [build 4/4] RUN mvn -f /home/app/pom.xml clean package                                                            276.2s
 => [stage-1 2/2] COPY --from=build /home/app/target/*.jar app.jar                                                      0.3s
 => exporting to image                                                                                                  0.3s
 => => exporting layers                                                                                                 0.3s
 => => writing image sha256:f654a42f84dd79517c6cbe43685c2b00a5feae3f94300d1a6ed5e84320c66bf4                            0.0s
 => => naming to docker.io/sravtar/currency-conversion:0.0.1-RELEASE                                                    0.0s
```

Run microservices as docker container :

```
ubuntu@ip-172-31-38-131:~$ sudo docker run -d -p 5000:8000 --name=currency-exchange sravtar/currency-exchange:0.0.1-RELEASE
e87e422f7a899de51765ea749a3ce5070aa4153838fe554fe6963dd12e096d7b
```

```
ubuntu@ip-172-31-38-131:~$ sudo docker run -d -p 9000:8000 --name=currency-conversion sravtar/currency-conversion:0.0.1-RELEASE
6129374dfad2b6d3bfd23ad1ab82d4f925dc689a30daf862661b9736befb6faa
```

Both container running .

Try accessing microservices .

Not secure | 13.235.241.27:5000/currency-exchange/from/USD/to/INR

```
// 20230413150635
// http://13.235.241.27:5000/currency-exchange/from/USD/to/INR

{
  "id": 10001,
  "from": "USD",
  "to": "INR",
  "conversionMultiple": 65.00,
  "exchangeEnvironmentInfo": "e87e422f7a89 v1 f7a89"
}
```

Microservice on 5000 port is accessible .

In docker default networking mode is bridge network . Containers, which are present in the default bridge network cannot directly talk to each other using local host.

```
 connect: permission denied
ubuntu@ip-172-31-38-131:~$ sudo docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
f8c25ada8f1b   bridge    bridge    local
a36e517926af   host      host      local
aa3f5e8504f0   none      null      local
ubuntu@ip-172-31-38-131:~$ sudo docker inspect network bridge
[
```

When we inspect the network , we can see that both container are part of this bridge network .

```
        "ConfigOnly": false,
        "Containers": {
            "330ee592731600ad931bc8d2aa80d699306def4259f74b076b95257bf4a8edfc": {
                "Name": "currency-exchange",
                "EndpointID": "7210bb98c0f8c1e3c9a480878d84e9cca648c2b4d244bd09fc22da3c4b1a3352",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            },
            "fb5c101c86ca7831cfd2a13208ddfce5753ff9fb899600f0d23a5a5429333345": {
                "Name": "currency-conversion",
                "EndpointID": "4c6c2b3050e50c651ddd7f08af8e685ad2a04a5ea6d716445e29ec630a715fbb",
                "MacAddress": "02:42:ac:11:00:03",
                "IPv4Address": "172.17.0.3/16",
                "IPv6Address": ""
            }
```

When containers are part of bridge network they cant communicate to each other .
We want both microservices to communicate with each other , so we will create a link from conversion service to exchange service .

We also have to specify url for exchange servce ,
Conversion service is depended on exchange service . so in conversion service link exchange service and specify url as exchange service .
sudo docker run -d -p 8100:8000 --env CURRENCY_EXCHANGE_URI=http://currency-exchange:8000 --name=currency-conversion --link currency-exchange sravtar/currency-conversion:0.0.1-RELEASE

```
ubuntu@ip-172-31-38-131:~$ sudo docker run -d -p 8100:8000 --env CURRENCY_EXCHANGE_URI=http://currency-exchange:8000 --name=currency-conversion --link currency-exchange sravtar/currency-con
version:0.0.1-RELEASE
dc75f1343eee744cd9f6a31e8ae490f4182ccaa6ccf5fab56e250af4ffaf3c07
docker: Error response from daemon: Cannot link to a non running container: /currency-exchange AS /currency-conversion/currency-exchange.
ubuntu@ip-172-31-38-131:~$ sudo docker start currency-exchange
currency-exchange
ubuntu@ip-172-31-38-131:~$ sudo docker run -d -p 8100:8000 --env CURRENCY_EXCHANGE_URI=http://currency-exchange:8000 --name=currency-conversion --link currency-exchange sravtar/currency-con
version:0.0.1-RELEASE
docker: Error response from daemon: Conflict. The container name "/currency-conversion" is already in use by container "dc75f1343eee744cd9f6a31e8ae490f4182ccaa6ccf5fab56e250af4ffaf3c07". Yo
u have to remove (or rename) that container to be able to reuse that name.
See 'docker run --help'.
ubuntu@ip-172-31-38-131:~$ sudo docker rm dc75f1343
dc75f1343
ubuntu@ip-172-31-38-131:~$ sudo docker run -d -p 8100:8000 --env CURRENCY_EXCHANGE_URI=http://currency-exchange:8000 --name=currency-conversion --link currency-exchange sravtar/currency-con
version:0.0.1-RELEASE
1f81328f56c2729a33278376367a04f37db11131ccb4b41d5199a6b99ee10f47
```

Use custom network to connect microservices :
Host network will connect the container directly to host . So the container will not use any network of its own. It will directly run on the host network itself. So if you are having a container which would run on port 8,000, then it's directly exposed onto the host. So that's one option we can use.The other option, which is present in here, is none.So none means no networking at all. So we'll not have any connection to any other containers. So we cannot use none to connect the microservices.

We can also create custom network .We can create network for our currency services alone .

Stop the containers and launch services as part of new network .
Create both container as part of new network .
default networking mode in Docker is bridge network. when the microservices are launchedinto the default bridge network they cannot directly talk to each other. And to help them to talk to each other we had to establish a link between them. We saw the Docker offered two other networking options, none and host. None means no networking. So we cannot have none as the networking mode for these microservices because we want them to talk to each other. And host networking only works on Linux. And when you deploy the containers on the cloud we created a custom network. We created a network called currency network and we launched up both the microservices in the currency network and got them talking to each other. One of the things you would've already observed is the fact that these commands to launch the containers are becoming huge. So to launch the two microservices we have two long commands. Now, if I have 5 or 10 microservices, how do I launch them up? How do I get them to talk to each other? It becomes very, very difficult. Let's see how to solve it in the next steps.

Using Docker compose :

If you go to the homepage of the Docker Compose you just need to do a Google for Docker Compose and you'd be able to land up on this particular page. And you can see that Docker Compose is a tool for defining and running multi container docker application. So when you have multiple containers and you want them to talk to each other, you can use Docker Compose. You can specify the configuration in a simple YAML file and then you can launch up the entire configuration with multiple microservices with just one command. Now before we get started with Docker Compose you'd obviously need to install it. So you can go to install Compose. The great news about Docker Compose is if you're using Docker desktop, Docker Compose is already present for you. So if you're using Docker desktop for Mac or Windows then the Docker Compose is already present as part of the installation. However, if you're on Linux or you are not using Docker desktop then you can follow the installation instructions which are present in here to install Docker Compose. What you have done is we have already created our Docker Compose file in the microservices folder. So in the microservices folder, if you open up the Docker Compose dot YAML file,

Install docker compose .

sudo apt update
sudo apt install curl unzip
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose
docker-compose --version

docker-compose up : to create microservices and network specified in docker compose file .

```
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/microservices$ cat docker-compose.yml
version: '3.7'
services:
  currency-exchange:
    image: sravtar/currency-exchange:0.0.1-RELEASE
    ports:
      - "8000:8000"
    restart: always
    networks:
      - currency-compose-network

  currency-conversion:
    image: sravtar/currency-conversion:0.0.1-RELEASE
    ports:
      - "8100:8100"
    restart: always
    environment:
      CURRENCY_EXCHANGE_SERVICE_HOST: http://currency-exchange
    depends_on:
      - currency-exchange
    networks:
      - currency-compose-network

# Networks to be created to facilitate communication between containers
networks:
  currency-compose-network:
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/microservices$
```

docker-compose up -d : to start containers in detached mode .
docker-compose down : to stop all services .

```
networks:
  currency-compose-network:
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/microservices$ sudo docker-compose up -d
Starting microservices_currency-exchange_1 ... done
Starting microservices_currency-conversion_1 ... done
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/microservices$ sudo docker netowkr ls
docker: 'netowkr' is not a docker command.
See 'docker --help'
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/microservices$ sudo docker network ls
NETWORK ID     NAME                                   DRIVER    SCOPE
41913e2ac867   bridge                                 bridge    local
3034c3dcea83   currency-network                       bridge    local
a36e517926af   host                                   host      local
64d40f305c68   microservices_currency-compose-network bridge    local
aa3f5e8504f0   none                                   null      local
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/microservices$ sudo docker-compose down
Stopping microservices_currency-conversion_1 ... done
Stopping microservices_currency-exchange_1   ... done
Removing microservices_currency-conversion_1 ... done
Removing microservices_currency-exchange_1   ... done
Removing network microservices_currency-compose-network
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/microservices$
```

docker system prune -a : to remove all services which are not being used .

```
Removing network microservices_currency-compose-network
ubuntu@ip-172-31-38-131:~/devops-master-class/projects/microservices$ sudo docker system prune -a
WARNING! This will remove:
  - all stopped containers
  - all networks not used by at least one container
  - all images without at least one container associated to them
  - all build cache

Are you sure you want to continue? [y/N] y
Deleted Containers:
4ffa08c80f1894f81b891b0797bd5c5450a575181082b8f81e9e29122c8e7dfc
274c671a28d408fb729467a646935b7b144b82d80aed19996e8cfc520579feb5
4e70858b3ae759c3b8649ca0ab359775b95f86a961bb8118dbf3ad81d5c5ecbb
562541812fd29420ac9ea81a065840a075547dd41c9b49a668a440f4bafe176d
db0d4f6f656f7dd15ff22c2c40f76b634bdd37658c005d77ea8cb3b4660c2a48
c7a846301a7bb1f140679d8ea88316802553e7a5810543fd5929939969c98df0
9f86c53693afaa8d1bb2ea1895f40795dc21f7d96e5b1b0384c782ba2314220f
b2075a3d177f8f691a8691fe3f84ba777bcae2ef43b7ef772fef38ffcad1c315
fcff9a5b616246a589b03388d47b19d2b941a2ae50d1471e187a903c2df5415d
81153c57757754e5e2914c52a1ca987c2a58d3f94f0fd791eb1fef0397b239c4
87d818210e42ac6aca37663552a583db929ac6d96865ed808f37826f0e0dc345
3eebb846443a4e83e27b1886a1f2ca8ecb821e212bb2adb44d9a19979e8fb3a5
156a4f961cd13305bd49aa0c97eba3d5a52acbc7876217af148ce9d2ddca0988

Deleted Networks:
currency-network
```

docker-compose events : to check all events happening .
So, "docker compose config." It would show the configuration which is used to launch the Docker Compose. So, you can also do a "docker compose images," similar to "docker images." It gives you the list of images which are being used by Docker Compose.

You can do a "docker compose ps," to list down the containers. You can do a "docker compose top," similar to "docker top," you can do a "docker compose top." It would give you the top processes which are running in each of the containers. So, currency conversion, which is the top process, currency exchange, which is the top process. You can also pause, and unpause, all the composed containers at the same time. So, you can say "docker compose pause," or you can do "docker compose unpause," similar to the "docker pause" command. And, you can also do "docker compose stop." Similar to "docker stop," you can also do a "docker compose stop." Or, you can do a "docker compose kill." So, these are very, very similar to the stop and kill commands that we used earlier.