Page :
Date :

Docker Definitions.

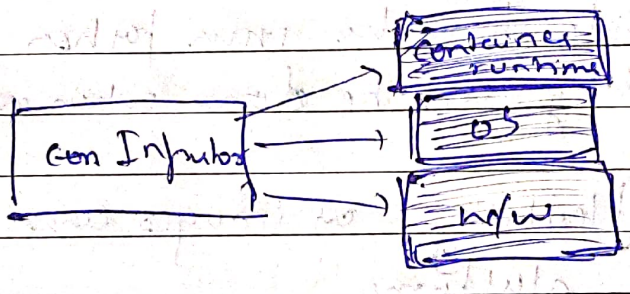
(1) provide tooling platform to manage the lifecycle of Containers.

(2) container became unit for building & testing for application distributing.

(3) when we are ready to depoly applic. in production environment. It works same whether your ~~deploy in~~ production environment is a local data center, cloud provider, hybrid of two.

(4) Docker became popular with developers because of its simple architecture, massive scaling & portability on multiple platform, environment & locations

(5) Docker isolates app. from infrastructure, hardware OS and contains runtime



(6) Written in Go programming & uses Linux kernel features to deliver its functionality.

(7) Docker uses namespaces to provide an isolated workspace. called container.

It Set of namespaces of each container and each aspect Create in seperate namespace. with access limited to that namespace.
Run

Camlin

Page :
Date :

* Docker offers following benefits.

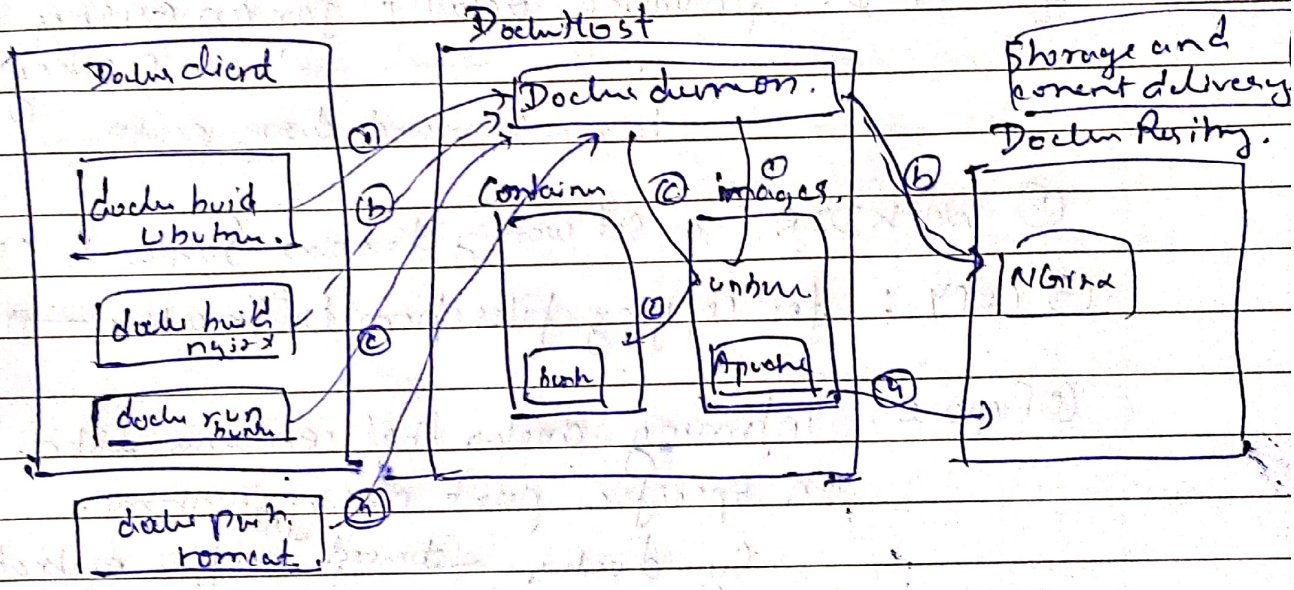
(1) Isolated and consistent env. result in stable appli. deployment.
(2) deployment occurs in sec. as docker images are small, reusable, can be in multiple projects.
(3) docker automation help eliminate errors & maintenance cycl.
(4) docker supports agile CI/CD devops practices.
(5) docker easy versioning speed up testing, ~~learning~~ and rollbacks. rep redeployment.
(6) docker helps segment application for easy cleanup, refresh & repair.
(7) developer collaborate to resolve issues faster.
(8) on demand scaling can be done when needed.
(9) provide greater flexibility as images are ~~portable~~ portable across platforms.

less down time & faster resolution of error.

Docker is not good for application that require high performance or security.
As these application are build on monolithic architecture.

Docker architecture

1. Docker client.
2. Docker daemon.
3. Communication
4. ~~API Req~~ Rest API
5. Docker management Lifecycle.
6. Images and Containers Handling.
7. Docker registry



Docker Registry: A Docker registry stores docker images. Docker hub is an public registry that any one use. by default Docker is configured to pull images from docker hub.

when we use `docker run <image-name>` command or `docker pull <image-name>`

Docker Hub: → Service provided by docker for finding and sharing container images.

Dockerfile (used to create own image).

A dockerfile is text file that used contains step-step instruction for creating Docker images.
we can docker file using any note editor editor from console or terminal.

Eg ① FROM ubuntu <base-image> → to pull images.

② Run: to execute command
execute any linux command
docker.

changes always made in docker file as any changes made in docker container can't persist.

③ CMD: providing default for an existing container executing
↓
to run command during execution

④ WORKDIR to set working directory similar to CD.

⑤ COPY: for copying files from local machine → host docker container.

⑥ Expose: informing docker that container listen on specific port during runtime.
So docker forward traffic on that port.

⑦ ENV: to set up environment variable.

Install tomcat on centos.

FROM centos:latest
RUN yum install java -y → install java.
RUN mkdir /opt/tomcat.
WORDIR /opt/tomcat
ADD <URL> → download tomcat files this url.
RUN tar -xvzf <filename> → directory name extract tar file
RUN mv <filename directory-name> /opt/tomcat. ← move the files.
EXPOSE 8080.
CMD ["/opt/tomcat/bin/<script-name>", "run"].

Docker Image

A docker images are read only template. with instructions for creating docker containers.

Each instruction in docker file will creates a layer in image.

image is based on another image with some customization.

New image = base-image + Customization

Eg we have to build an image that based on another ubuntu image, that install tomcat Apache Server. In it along with our application customization and env variables. for our application.

need to create. Dockerfile

⇒ each instruction in image file will create a layer.

⇒ and we after changing the Dockerfile, if we rebuilt the image only those layer are build that get changes. Rest previous layer image are copied from previous image.

That make docker image light weight, small size, fast and allow versionalization tech.

Then we use,

docker build -t <image-name> .

current directory

Page :
Date :

Containers

(1) Running instances of docker images.
(2) We can create, delete, start, stop our containers using command line interface or GUI.
(3) ~~We can create two or more~~ We can connect a container with two or more containers networks, attach storage to it.
(4) By default containers are isolated from other & host machine.
(5) We can control isolated container networks, storage or other underlay system from other containers & host machine.

* Images Naming
  Eg docker.io / ubuntu : 18.04

Page :
Date :

Bind Mounts

1) Limited functionality as compared to docker volume.
2) When bind mount is used file / directory on host machine is attached to container.
3) The file path or directory referenced as full path on host.
4) Bind mount created on demand if does not exist.
5) bind mount very performant, but they rely host machine file system, sp. follow specific directory path.

Bind mount Sensitive.

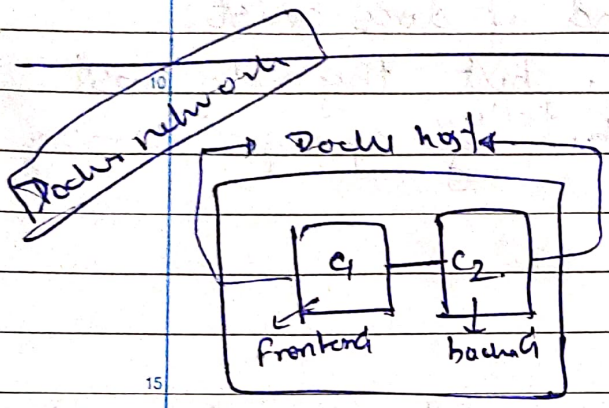
One of the side effect of using docker bind mount that it allow to alter the files and directory of host machine via process running on docker container
including deleting, updating, modifying, creating the files and directory on host machine.
This powerful ability to alter the host machine files have many security implication and effect including non docker process running on host machine.

Docker compose

tool for defining & running multi container app. with docker application.

Compose + yaml ⇒ configure your application.

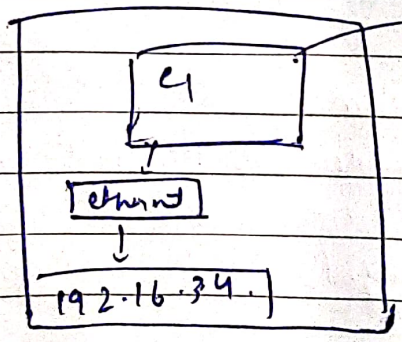
→ with single command we can start, stop all services from configuration.



There are two scenario's.

1. one container want to communicate with another container
2. one container want to completely isolate.

1. Container 1 —talks→ Container 2
2. Container 1 —isolates→ Container 2



→ container have different subnet and host have different subnet.

and how container talk to docker host by default using bridge ↓ virtual ethernet

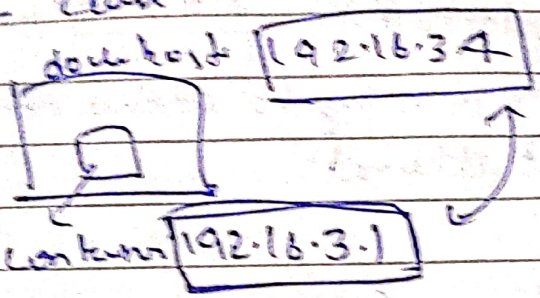
(start here if we remove default bridge

↳ Then our application is not accessible from outside world (not reachable)

↳ By default docker network is bridge network.

② **Host networking**: In host networking docker container directly use the ip address of host

host means docker host & network always belongs same class



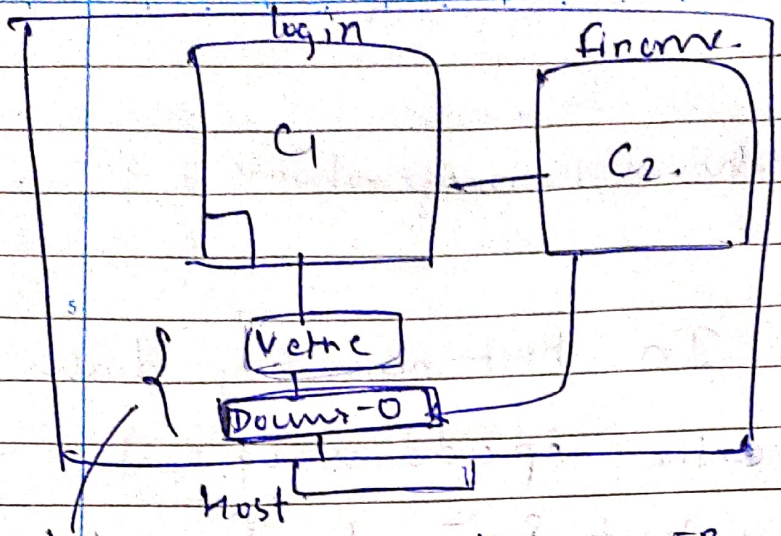
same class / CIDR network

both are in same subnet or same CIDR block.

∴ The docker host have directly access to docker containers.

③ **Overlay Networking**: useful when have multiple docker host.

② Scenario : One container should talk to another container.



both login & Finance uses the same virtual ethernet for taking the host
i.e, docker - 0
Common path for both

only one bridge ⇒ OOTB.
out of the box.

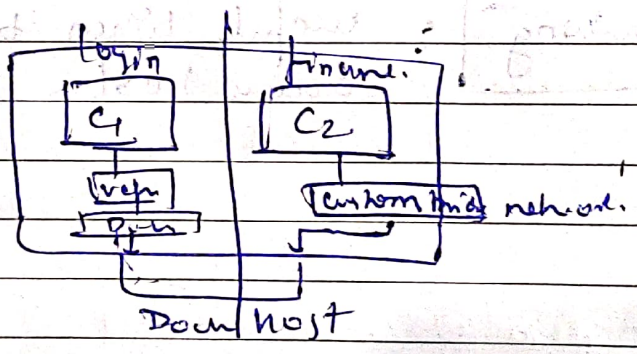
Not secure.

One container is talking to another container i.e. using default ethernet

③ Not talk $C_1 \xleftarrow{X} C_2$

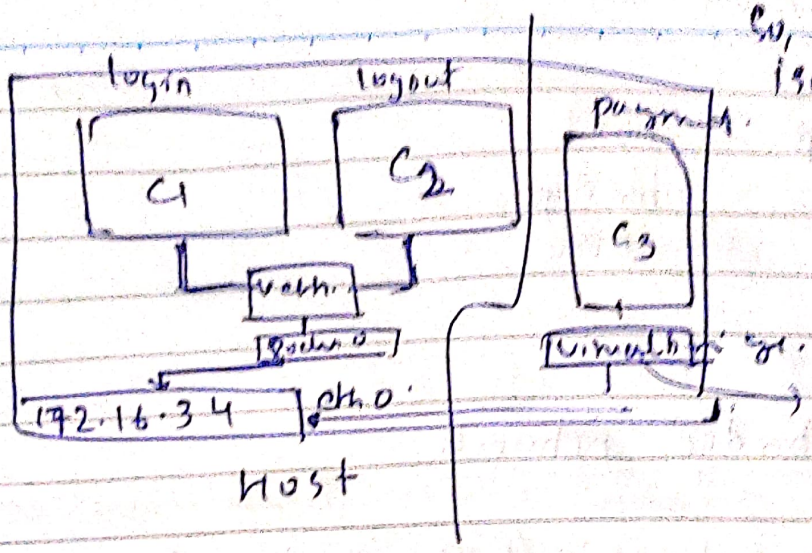
How to achieve isolation b/w C_1 and C_2 (i.e login & finance)

by creating custom / user defined bridge network.



By using custom bridge network we can achieve logical isolation b/w containers.

in above example we have created custom bridge network.



C_3, C_4, C_5 Completely isolated from C_1 & C_2.

To communicate b/w two containers C_1 & C_2, by default their is common network is Docker 0

From we can talk $C_1 \rightarrow C_2$.

Practical implementation (Connect ping core container from another).

① C_1: docker run -d --name login nginx

↳ get inside C_1 docker exec -it login /bin/bash

↳ install ping command. ⇒ apt-get install iputils-ping -y

② C_2: docker run -d --name logout nginx.

↳ get inside C_2

both C_1 & C_2 must be have same subnet as we use using default bridge. 172.17.0.2 (C_1) & 172.17.0.3 (C_2)

↓ default out of the box bridge network.

③ Now ping first container ⟷ from second container.

ping 172.17.0.1

- docker network ls.

↳ o/p host
bridge
none.

- custom bridge network

① create custom bridge network for finance network.

Step1: docker network ~~bridge~~ create secure-network } custom bridge is created

Step2: docker ~~network~~ network ls } To check whether bridge is created.

Step3: docker run -d ~~--network~~ --network secure-network ~~finance~~ --name finance nginx. }

Step4: docker ps } shows all running containers } ~~docker inspect~~ ps

① finance ② login ③ logout.

Step5: Now inspect our custom ~~network~~ container using.
docker inspect finance.

Step6: Now try to ping finance container ⟷ from login container

↓

This is how we make our containers secure using custom bridge network.

host container

docker run -d --name host-demo --network host nginx

The ip address of this host demo container is similar to docker host.

Here we are going to access host-demo container directly from host IP address.
it doesn't contain any custom IP.

attaching custom network to ~~image~~ container

Deploy Jungo application

Page :
Date :

requirement.txt Contain django.

Docker file

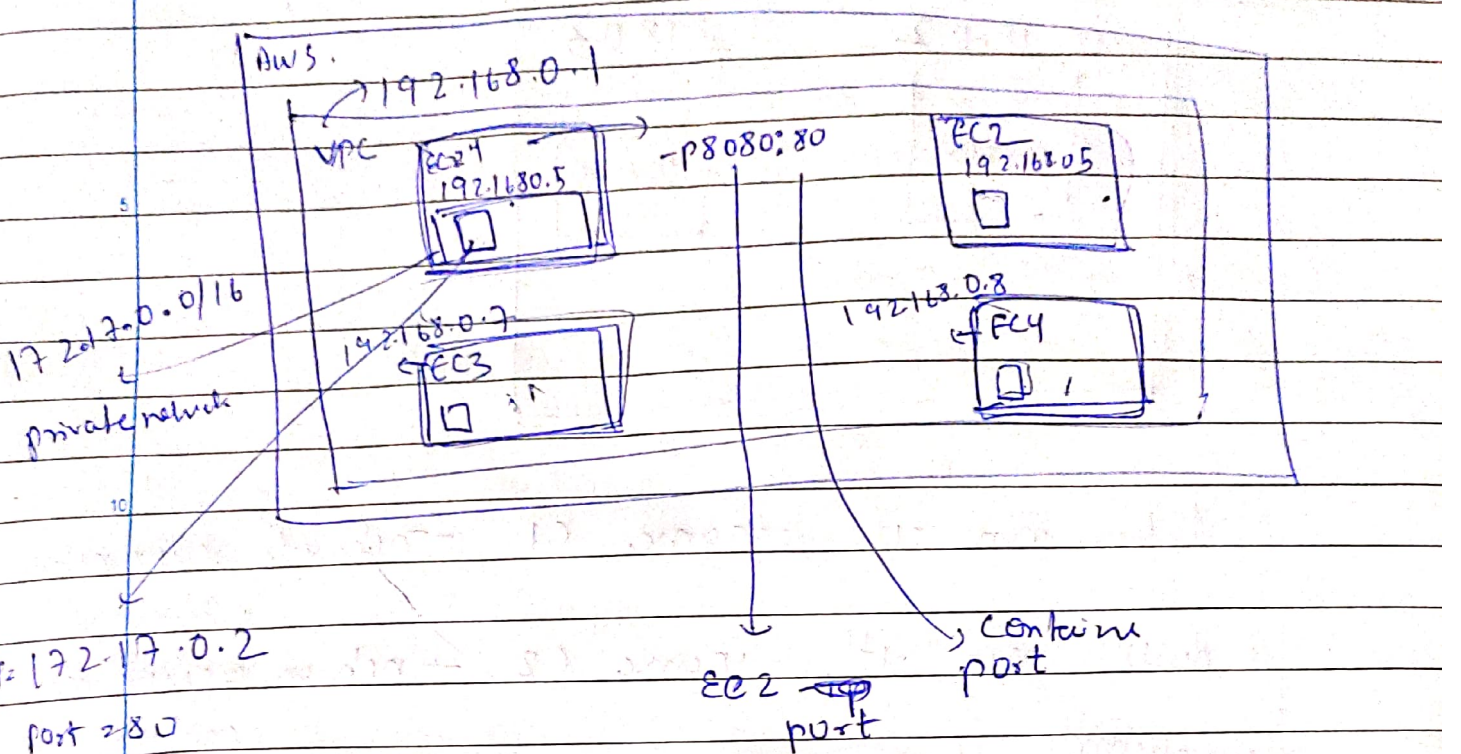
```
FROM ubuntu

WORDIR /app

COPY requirement.txt /app
COPY devops /app

RUN apt-get update && \
    apt-get install -y python3 && \
    pip install -r (requirements.txt) && \
    cd devops

ENTERY POINT ["python"]
CMD ["manage.py", "runserver"
     "python",]
```

Now docker build . } to build above dockerfile.

Now run container using.

docker run -p 8000:8000 -it <container name>

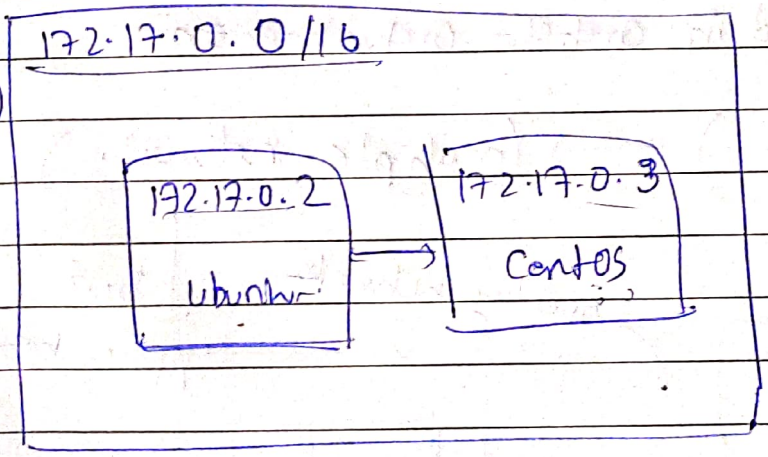
Overlay network



192.168.0.5:8080
= 172.17.0.2:80

default network

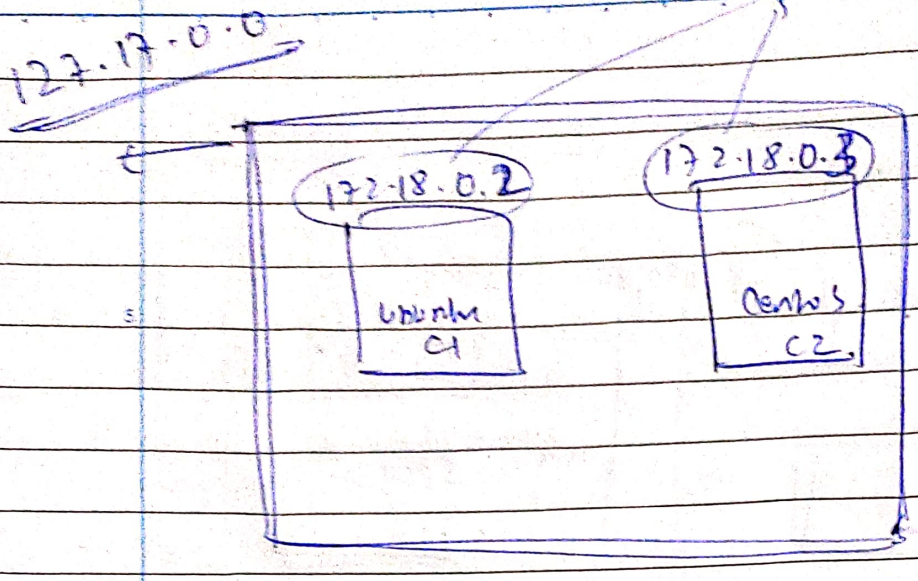
(private network)



How to go inside host's O/S

Ubuntu after running. docker exec -it <name> bin/bash

custom default user-defined bridge



docker run -it --name C1 --network secure-network ubuntu

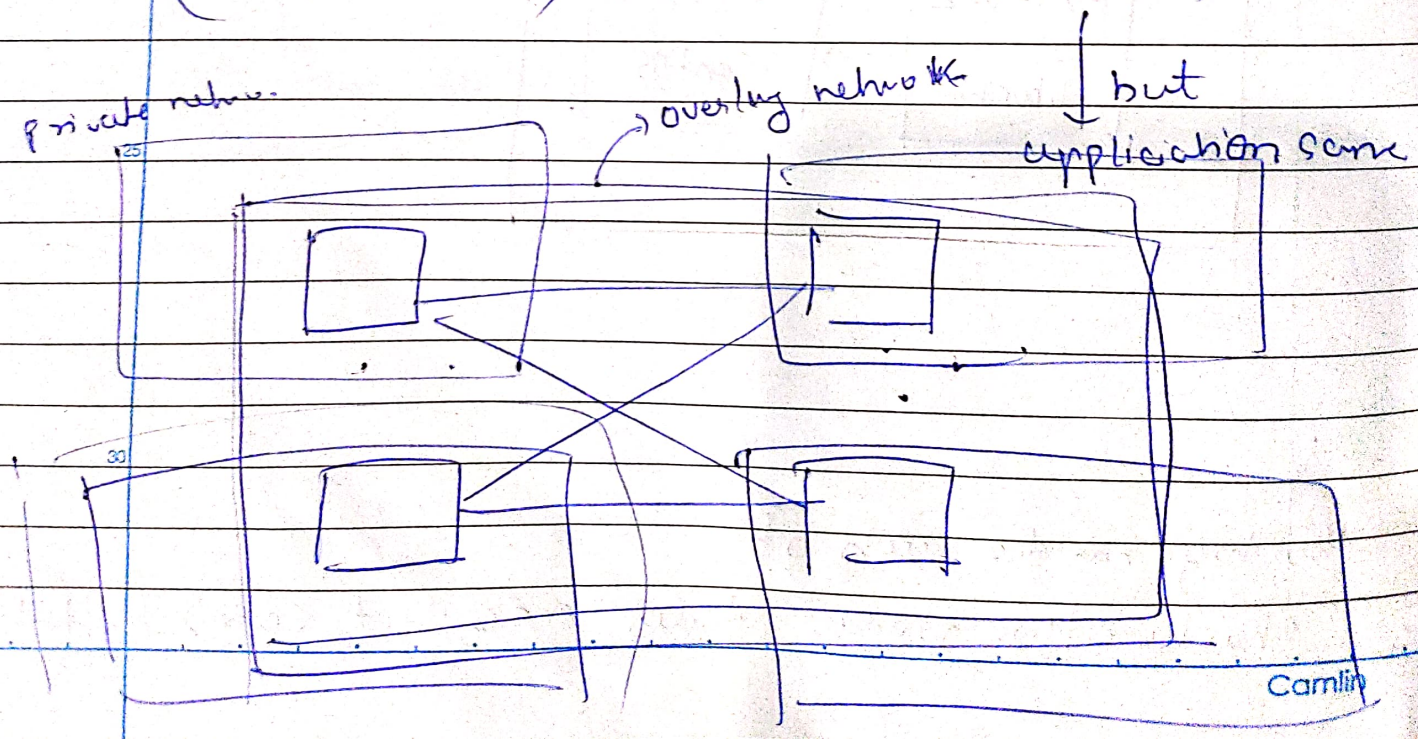
docker run -it --name C2 --network secure-network centos.

Advantages custom.

② C1 and can ping C2 with name as well ip address.

Overlay network : used in ~~Orchestration~~ Orchestration

(multiple-host) ⟶ (multiple-network)



javac -h .

1. cd dehub
2. mkdir javaapp
3. cd javaapp
4. touch javafile Hello.java → nano
5. nano Hellojava
6. touch Dockerfile
7. nano Dockerfile.

```
FROM openjdk:8.
COPY . /
WORKDIR /var/www/java
COPY . /var/www/java
RUN javac Hello.java.
CMD ["java", "Hello"]
```

} file

8. save Dockerfile.
9. Docker build -t ujeshImage . → Image.
10. Docker run -d -name ujeshName ujeshImage → Container
11. Docker push .

Camlin

Installing jdk.

by using apt-get install -y default-jdk

create java file from terminal.

Cat > Hello.java

```
class Hello{
    public static C.
}
}
```

Now type java Hello.java.

① docker login

② mkdir java-demo.

③ cd java-demo

④ touch Dockerfile. → WORKDIR /app1
From Ubuntu
Run apt-get install && apt-get install openjdk-11-jdk

⑤ docker build -t demoimage:1.0

⑥ docker push hitendra/demoimg COPY Hello.java /app1
RUN javac Hello.java
CMD ("java", "Hello")

⑦ docker pull hitendra/demoimage

⑧ docker run -d --name demoContainer hitendra/demoimg

--mount b. volume

volume

① docker volume create vol1

② docker run -d --name c1 --mount source=vol1 target=/app nginx:latest.

③ docker inspect

④ docker exec -it c1 bash

⑤ ls

⑥ cd app

⑦ notepad text.txt

⑧ exit

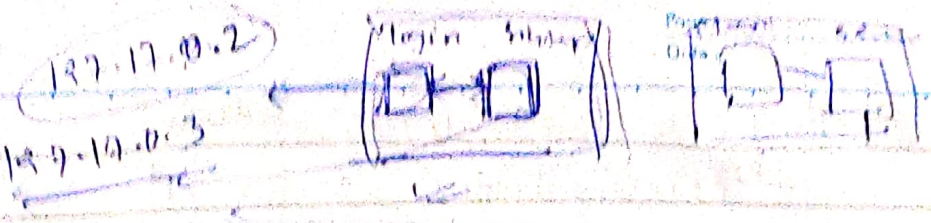
⑨ docker exec -it c1 bash

⑩ cd app

⑪ cat > text.txt.

bind

Exp - 5



(1) docker run -d --name (C1) dumoimage 1

(2) docker run -d --name C2 dumoimage 2

(3) docker inspect C1

(4) docker inspect C2

(5) docker exec -it C1 bash

(6) ~~docker ls~~

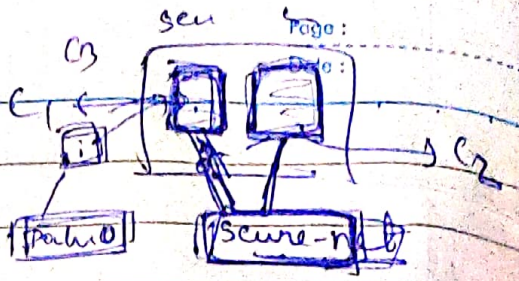
(7) ping 127.17.0.3

(8) exit

(9) docker exec -it C2 bash

(10) ping 127.17.0.2

(11) exit



① docker network creat ~~type~~ --driver=bridge secure-net

② docker run -d --name C1 img1

③ docker run -d --name C2 img2

④ docker network connect secure-net C1

⑤ docker network connut secure-net C2

⑥ docker inspet C1

Ubuntu #

⑦ docker inspet C2

⑧ docker exec -it C1 bash

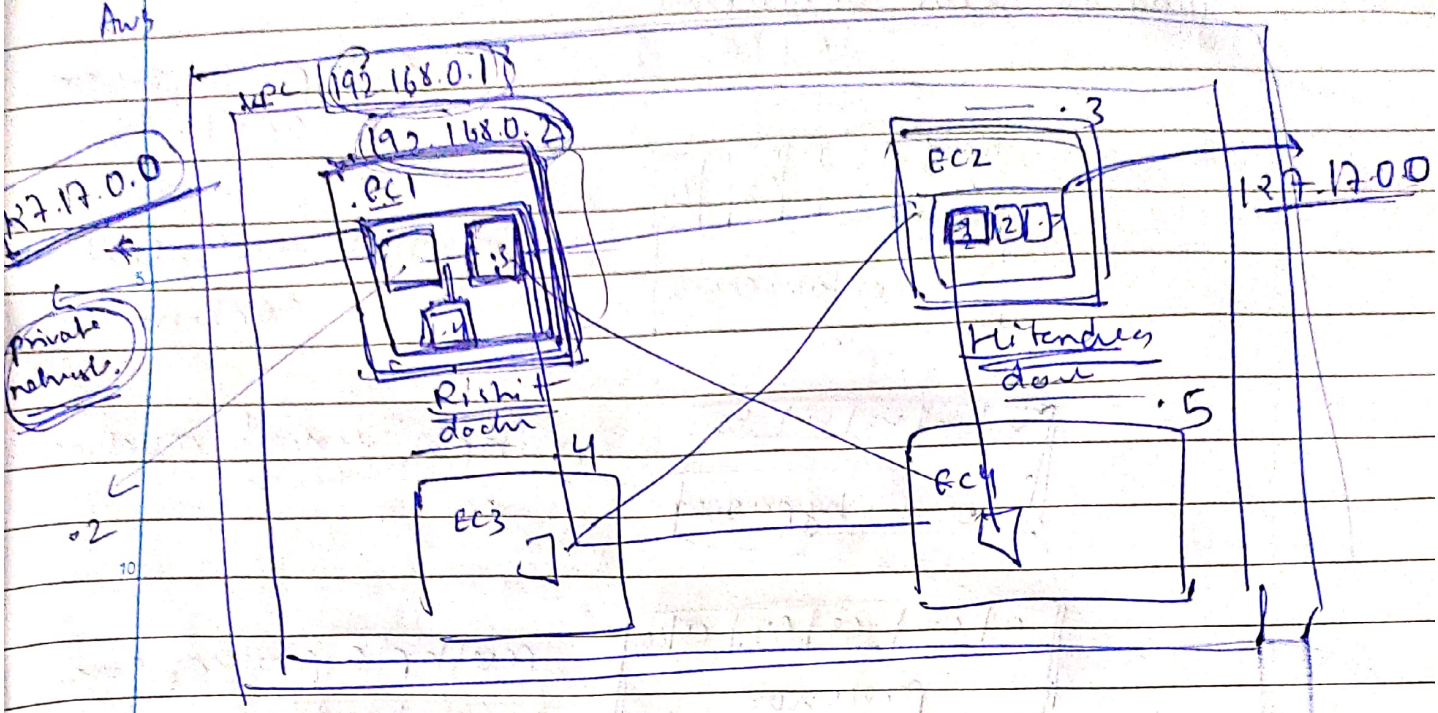
⑨ apt-get install iputils-ping

⑩ ping C2

⑪ exit

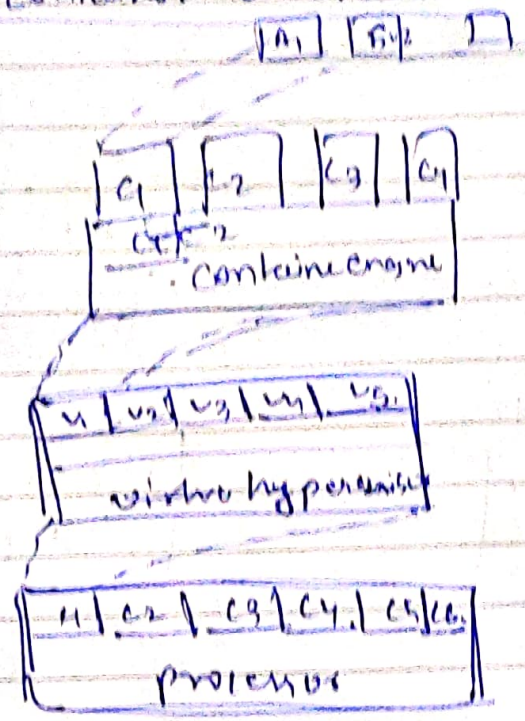
Bridge network

Page :
Date :

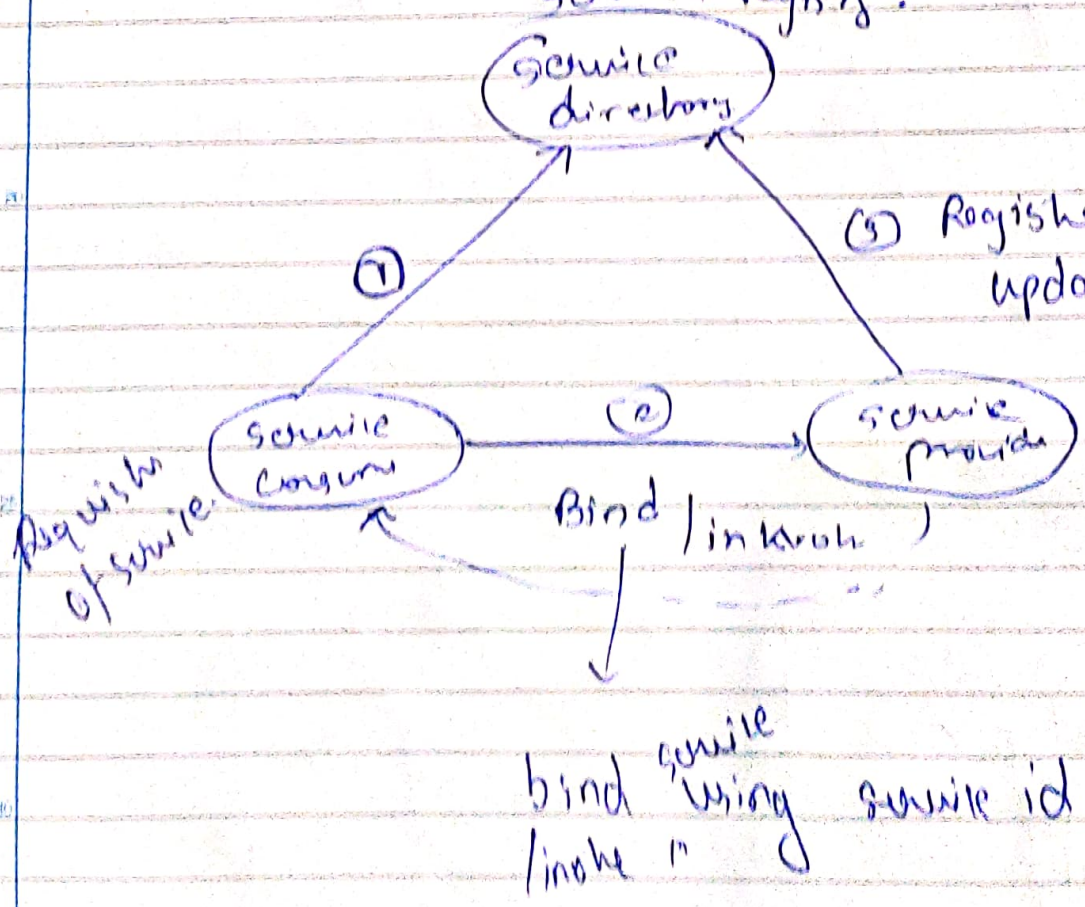


Page :
Date :

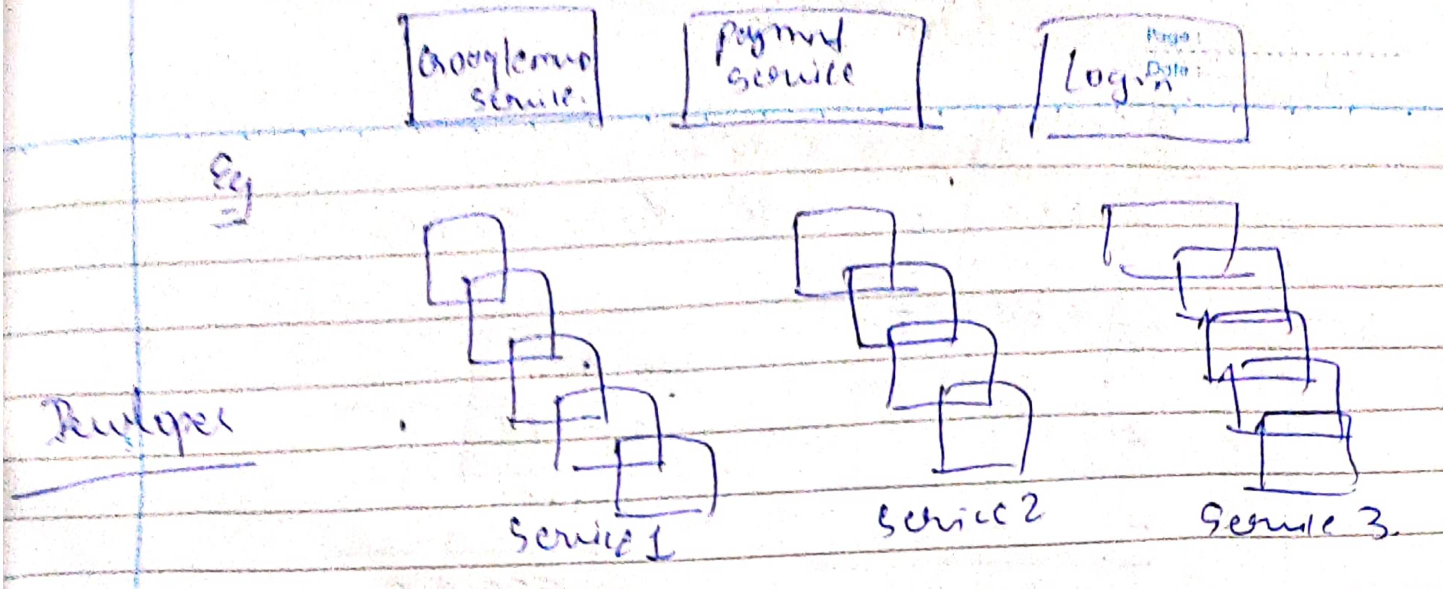
Hybrid container architecture



Service Oriented Architecture (SOA)



using.
WSDL pro
UDDI
API



Advantage
① Reusable components
② scalability
③ Resilient.

Disadv
① ~~more~~ complex ~~stack~~ design & imp
② more network traffic.

→ web service description language

→ Universal, description discovery, integration