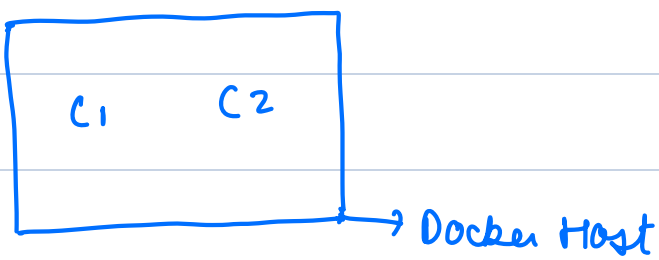
Docker Container Networking and Security Cont.

- starts at 9:05 pm

Agenda

1. Intro Docker Networking
2. Docker Networking commands
3. Types of Networks
4. Exposing containers externally
5. Network Troubleshooting
6. Configuring Docker to use External DNS

Docker Networking



C1 → C2

C1 → host

C1 → External network.

In Docker, networks are isolated environments

C1 → C2

C1 → C3 ✗

C3 → C4

Docker Networking Commands.

1. listing networks

   docker network ls

2. Inspecting network

   docker network inspect network_name

3. Creating a Network

   docker network create network_name.

④ Conneting a container to a network

docker network connect network_name cont_name

⑤ Disconnecting

" " disconnect " "

⑥ Removing a network

docker network rm network_name

⑦ Run a container with specific network

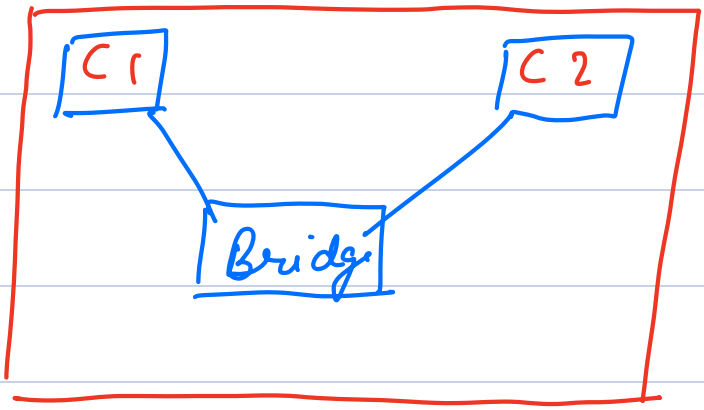
docker run --network network.name image

⑧ Pruning

docker network prune

Types of Network (Drivers)

① Bridge Network



containers on default bridge network can't communicate with hostname, but can communicate using IP address.

default bridge network doesn't include a dns to resolve container names.

Create 2 containers.

docker run -dit --name container1 alpine sh

docker run -dit --name container2 alpine sh

Check the IP address of the containers

docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' container1

docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' container2

Access container 1

docker exec -it container1 sh

ping IP

    You should see successful pings.

ping container2

    This will **fail**, as the default bridge network does not support DNS-based name resolution.

Create custom Bridge Network

Create a User Defined Bridge Network

docker network create my_bridge

Run Containers Using the user defined Bridge

docker run -dit --name container1 --network my_bridge alpine sh

docker run -dit --name container2 --network my_bridge alpine sh

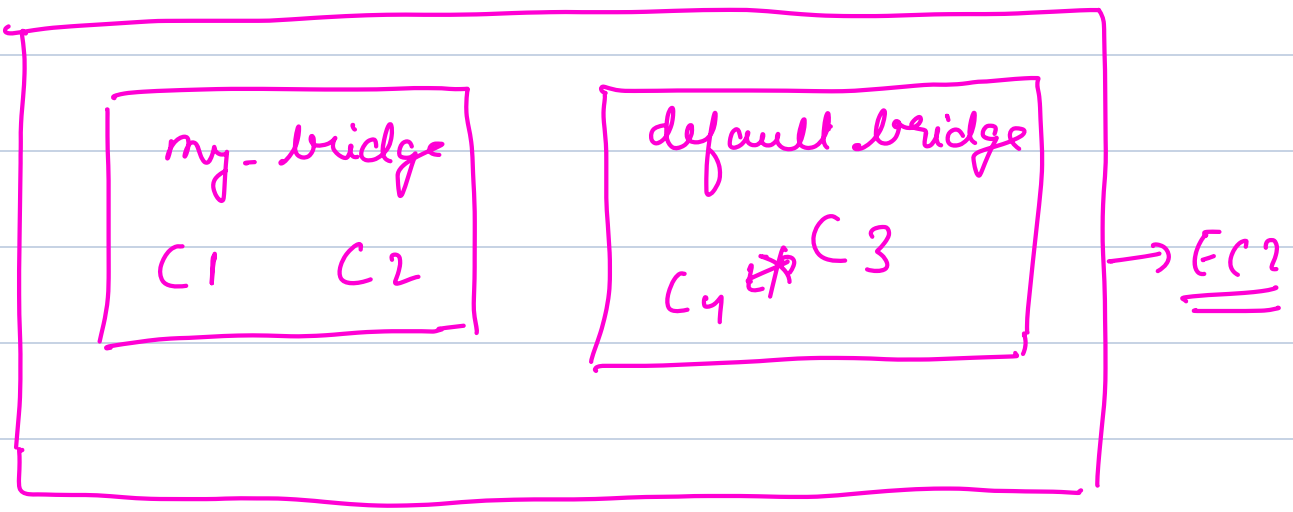
Can see container1 and container2 part of the network

docker network inspect my_bridge

Execute inside container1

docker exec -it container1 sh

ping container2



docker run -dit --name container3 alpine sh

get ip of C1

↓

execute into C3

↓

ping to C1 → fails

How to disable ICC on default Bridge

com.docker.network.bridge.enable_icc

↓

icc → inter container communication

false

`/etc/docker/daemon.json`

```
{
 "bridge": "docker0",
 "icc": false
}
```

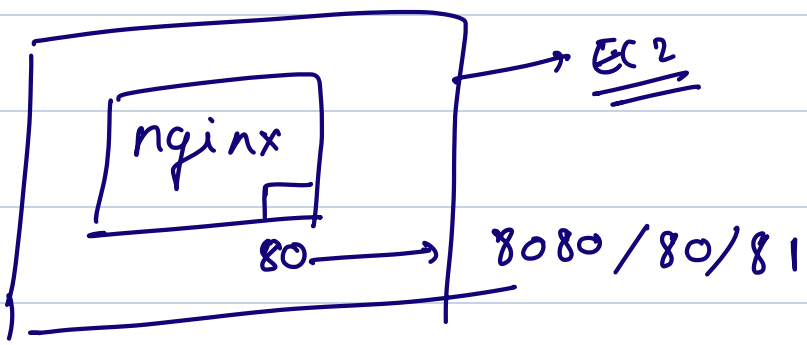
→ default docker network interface.

→ restart docker.

{ IP table } [Firewall]

```
sudo iptables -I DOCKER-USER -i docker0 -o docker0 -j DROP
```

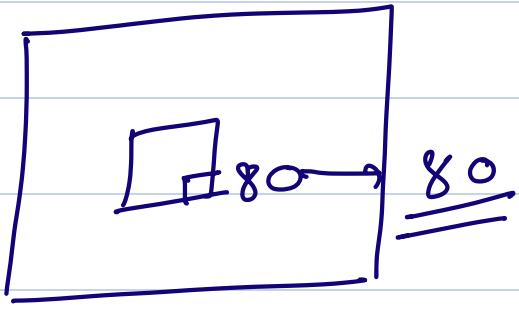
② Host Network.



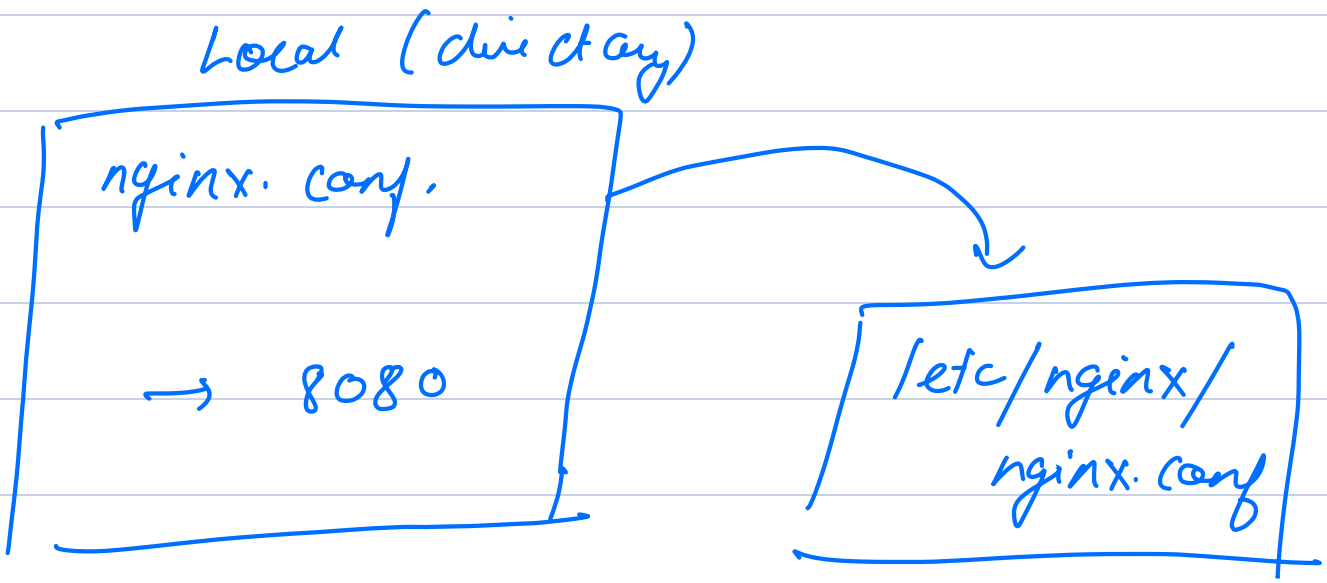
curl (EC2 IP) : 8080

↓

→ nginx out put.



→ Binds application to hosts port directly.



Break → 10:30 pm

docker run -d --network host --name host_net_demo -it nginx

curl localhost

Run multiple containers on same port.

```
docker run --rm --network host --name nginx1 -d nginx
```

```
docker run --rm --network host --name nginx2 -d nginx
```

→ not possible as port 80 already in use by nginx1

→ create 2 nginx containers using custom nginx.conf file.

↓

uses port 8080

nginx.conf

```
worker_processes 1;

events {
    worker_connections 1024;
}

http {
    server {
        listen 8080;
        server_name localhost;
        location / {
```

```
        root   /usr/share/nginx/html;

        index  index.html index.htm;

    }

  }

}
```

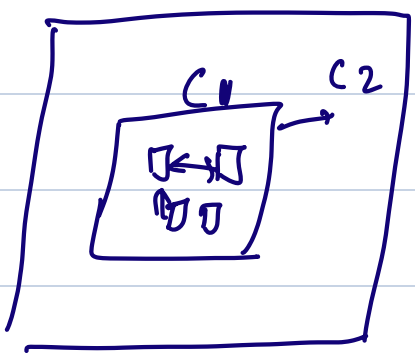
Bind mount.

```
docker run --network host -v $(pwd)/nginx.conf:/etc/nginx/nginx.conf:ro -d nginx
```

curl http://localhost:8080 → Served by custom nginx container

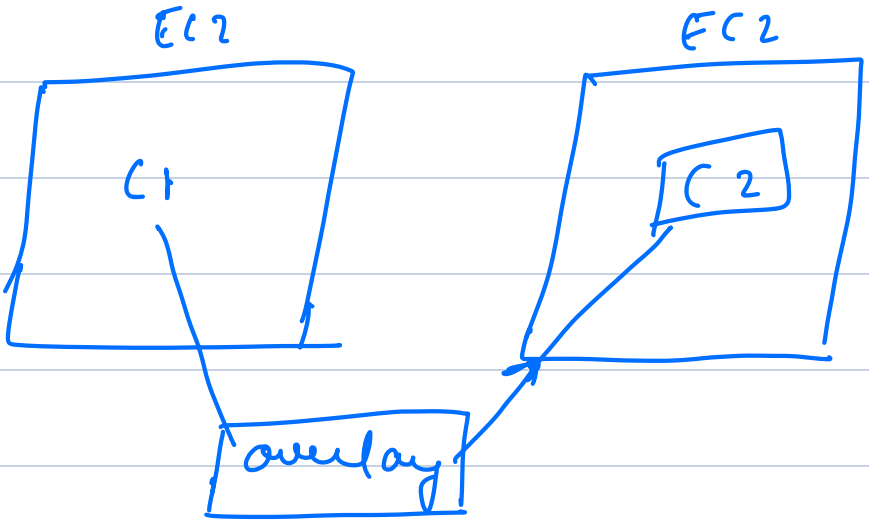
" :80 → " " nginx!

(3) None Network.

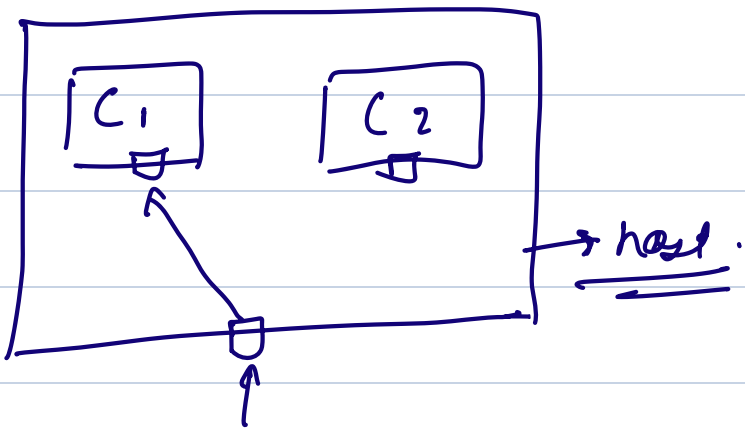


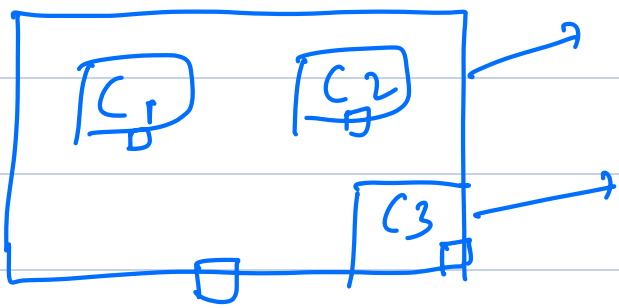

```
docker run -d --name isolated_task --network none my_data_image
```

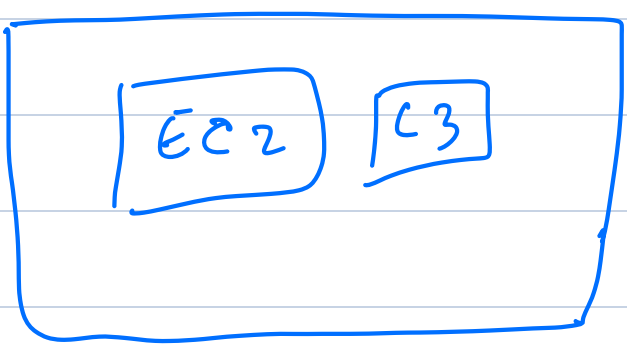
(4) Overlay Network.


EC2
EC2
C1
C2
overlay

→ MacVLAN


C1
C2
host.


EC2
C1
C2
C3
unique MAC address over IP address.


EC2
C3

```
docker network create -d macvlan \
  --subnet=192.168.1.0/24 \
  --gateway=192.168.1.1 \
  -o parent=eth0 my_macvlan_network
```

(6) → IPVLAN

```
docker network create -d ipvlan \
  --subnet=192.168.1.0/24 \
  --gateway=192.168.1.1 \
  --ipvlan-mode=l2 \
  -o parent=eth0 ipvlan_l2_network
```

→ same mac address of host.

Use Case 1: Cloud-Native Applications

- **Problem:** A company is deploying cloud-native microservices using Docker containers but needs to provide IP addresses to the containers that are routable from other services or the external network.

- **Solution:** **IPvlan** allows containers to be assigned IP addresses in the same subnet as the host, simplifying the network configuration for cloud-native applications and microservices.

→ How will you expose your containers externally ?

① Using port binding
-p --publish.

② docker-compose

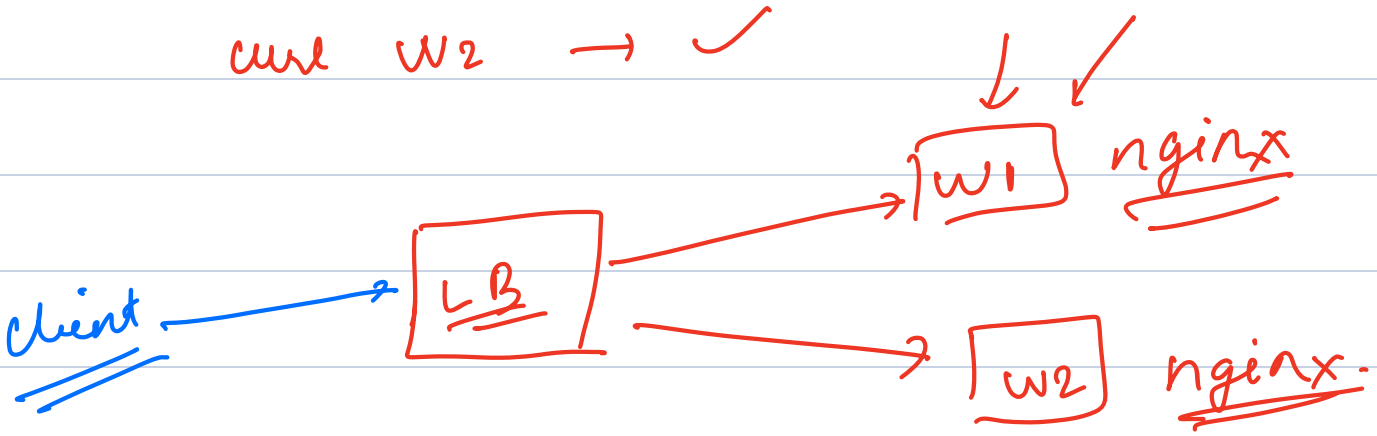
```
version: '3'

services:

 web:

  image: nginx

  ports:

   - "80:80"
```

③ In swarm cluster.

Servic -p 80:80

curl W1 → ✓

curl W2 → ✓



```
events {}

http {
    upstream backend {
        server 172.31.25.125:80;  # IP of Worker Node 1
        server 172.31.25.126:80;  # IP of Worker Node 2
    }
    server {
        listen 80;  # Port on which NGINX listens for external traffic
        location / {
            proxy_pass http://backend;  # Forward request to upstream backend servers (worker nodes)
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
    }
}
```

Network Troubleshooting

Tools and Techniques for Troubleshooting Docker Networks

Inspect Network Configuration:

- Use `docker network inspect` to view detailed information about the network configuration.

Check Container Connectivity:

- Use `docker exec` to run network troubleshooting commands inside a container.

- Execute a ping command inside a container

docker exec -it container1 ping container2

Verify DNS Configuration:

- Check DNS settings inside the container.

docker exec -it container1 cat /etc/resolv.conf

Why 127.0.0.11?

- Docker assigns `127.0.0.11` as a special IP for its internal DNS server.

- This DNS server is responsible for resolving:

  - **Container service names**: It resolves names of services in the same Docker network (e.g., `my-service`).

  - **External domain names**: It forwards requests for domains like `google.com` to the external DNS server

configured on the Docker host.

Port Binding Issues:

- Use `netstat` or `ss` to check port bindings on the Docker host.

- netstat -tuln

Use `docker logs`:

- Check container logs for any network-related error messages.

- docker logs container

Check Firewall Rules:

- Ensure that firewall rules are not blocking Docker network traffic.

iptables -L -n

Configuring Docker to use External DNS Server

--dns flag.

docker run -d --name my_container --dns 8.8.8.8 nginx

/etc/docker/daemon.json

```
{
  "dns": ["8.8.8.8"]
}
```

→ restart