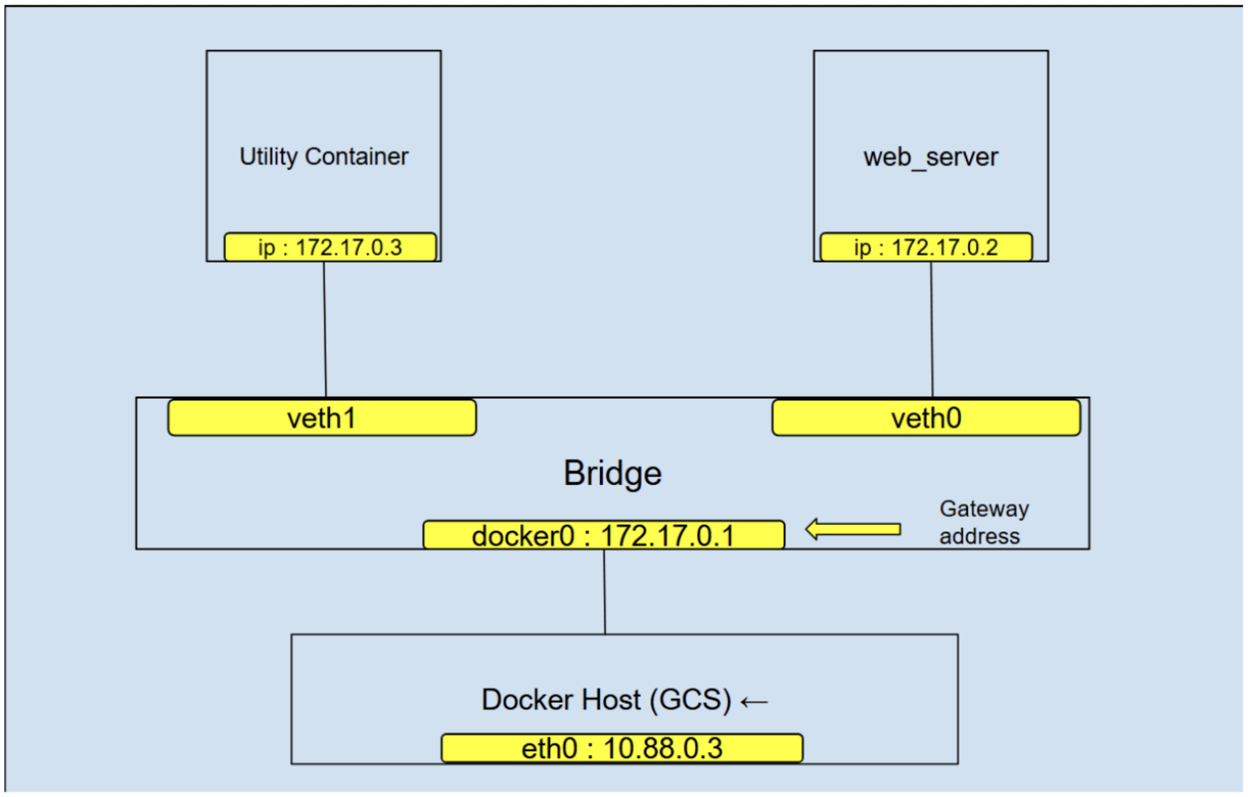
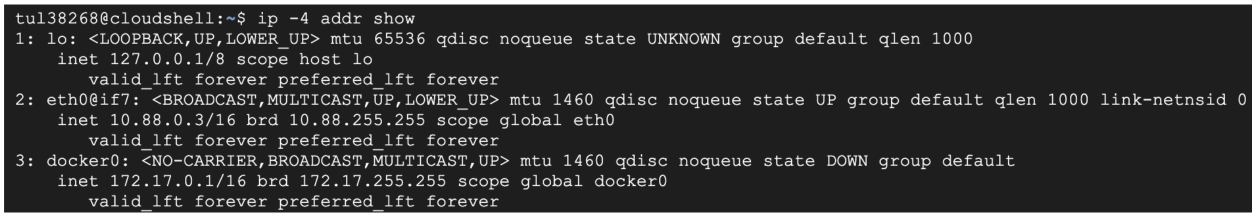
PART 1

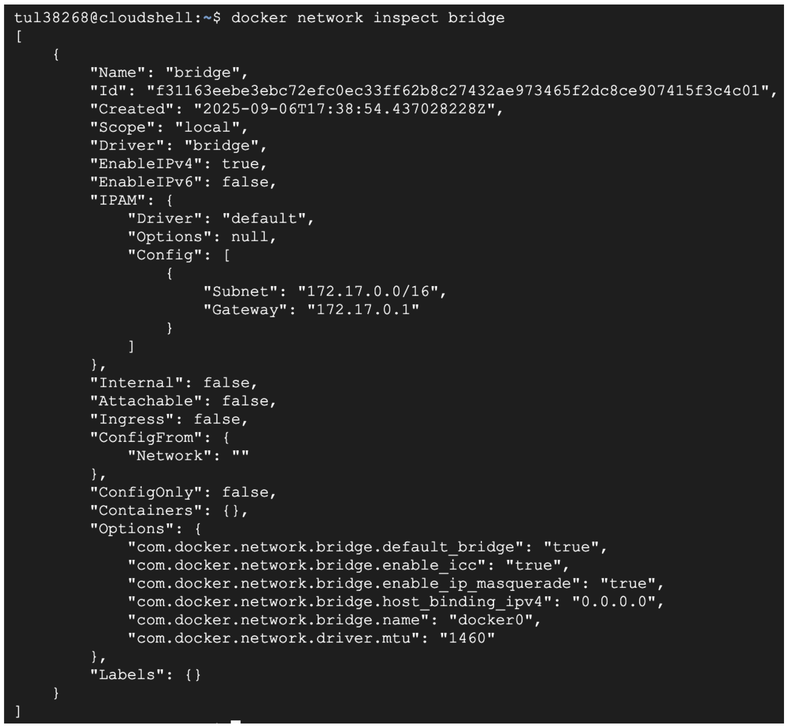


The bridge network in Docker acts like a virtual switch that connects containers to each other and to the host. Each container gets its own private IP address within the bridge subnet, while the bridge gateway (usually 172.17.0.1) serves as the router that links containers back to the host and beyond to external networks. When a container sends traffic, it goes through the bridge to the gateway, which then forwards it to the right destination. In my tests, both the NGINX and Ubuntu containers were able to communicate with each other using their internal IPs, showing that the bridge handles container-to-container traffic seamlessly. The setup also confirmed that without port mapping, services like NGINX remain reachable only from within the bridge network, not from the outside world.



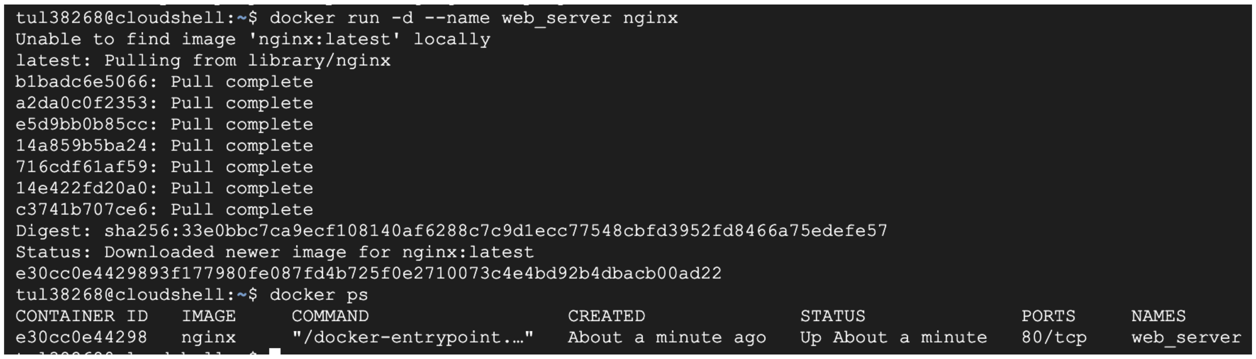
```
tul38268@cloudshell:~$ ip -4 addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: eth0@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc noqueue state UP group default qlen 1000 link-netnsid 0
    inet 10.88.0.3/16 brd 10.88.255.255 scope global eth0
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1460 qdisc noqueue state DOWN group default
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
```

Here in this screenshot we see all of the iPV4 addresses configured on the host's network interfaces. You also see the host ip address along with the subnet mask.



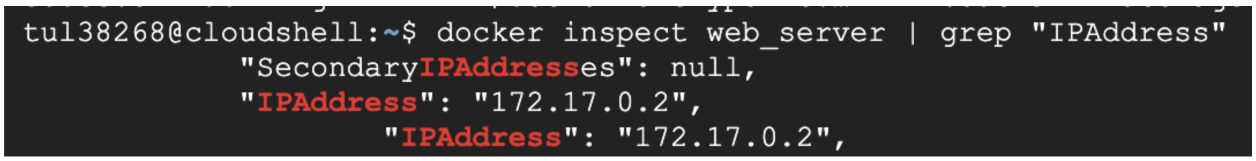
```
tul38268@cloudshell:~$ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "f31163eebe3ebc72efc0ec33ff62b8c27432ae973465f2dc8ce907415f3c4c01",
        "Created": "2025-09-06T17:38:54.437028228Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv4": true,
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1460"
        },
        "Labels": {}
    }
]
```

The bridge network is the default network Docker creates for containers. It gives each container a private IP address from its subnet and uses the gateway address (172.17.0.1) to let containers talk to the host system and reach the internet.



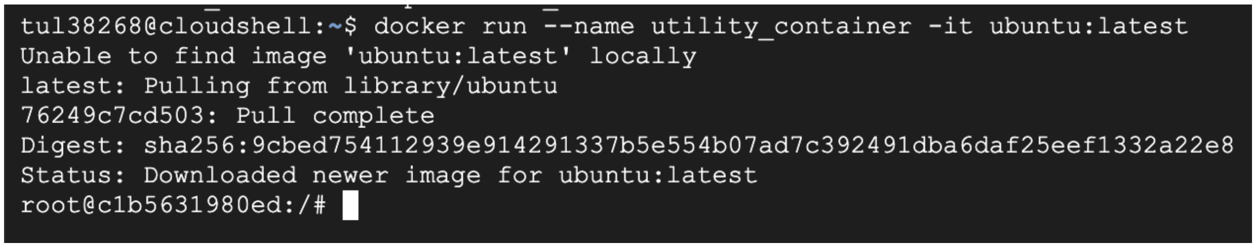
```
tul38268@cloudshell:~$ docker run -d --name web_server nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
b1badc6e5066: Pull complete
a2da0c0f2353: Pull complete
e5d9bb0b85cc: Pull complete
14a859b5ba24: Pull complete
716cdf61af59: Pull complete
14e422fd20a0: Pull complete
c3741b707ce6: Pull complete
Digest: sha256:33e0bbc7ca9ecf108140af6288c7c9d1ecc77548cbfd3952fd8466a75edefe57
Status: Downloaded newer image for nginx:latest
e30cc0e4429893f177980fe087fd4b725f0e2710073c4e4bd92b4dbacb00ad22
tul38268@cloudshell:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS              PORTS     NAMES
e30cc0e44298   nginx     "/docker-entrypoint.…"   About a minute ago   Up About a minute   80/tcp    web_server
```

This is me just running the nginx container and the docker ps command is further confirmation it is up and running.



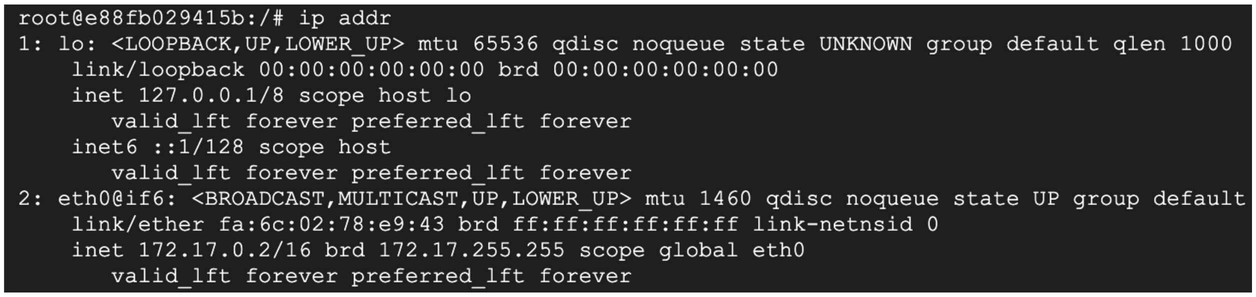
```
tul38268@cloudshell:~$ docker inspect web_server | grep "IPAddress"
            "SecondaryIPAddresses": null,
            "IPAddress": "172.17.0.2",
                    "IPAddress": "172.17.0.2",
```

This command shows the IP address assigned to the NGINX container on the Docker bridge network. This is the address other containers use to communicate with it, and it confirms the container is connected to the network.



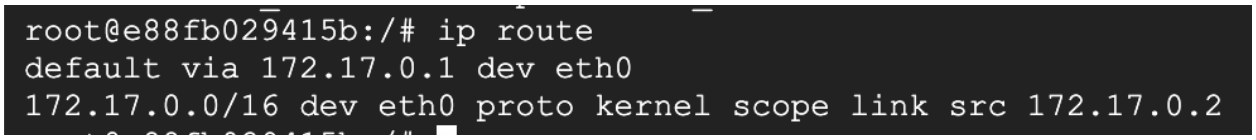
```
tul38268@cloudshell:~$ docker run --name utility_container -it ubuntu:latest
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
76249c7cd503: Pull complete
Digest: sha256:9cbed754112939e914291337b5e554b07ad7c392491dba6daf25eef1332a22e8
Status: Downloaded newer image for ubuntu:latest
root@c1b5631980ed:/#
```

Me running the ubuntu container.



```
root@e88fb029415b:/# ip addr
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: eth0@if6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc noqueue state UP group default
    link/ether fa:6c:02:78:e9:43 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
```

This shows the network interfaces in the ubuntu container and their respective assigned ip addresses.



```
root@e88fb029415b:/# ip route
default via 172.17.0.1 dev eth0
172.17.0.0/16 dev eth0 proto kernel scope link src 172.17.0.2
```

ip route displays the ubuntu container’s routing table, showing where network traffic goes; the important part is the default route pointing to the bridge gateway (usually 172.17.0.1), which lets the container reach the host and outside networks.

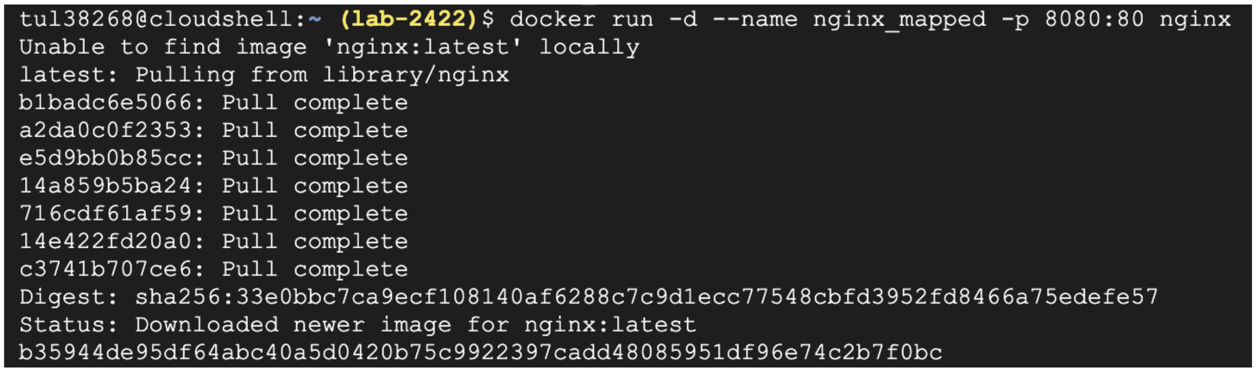
PART 2

1. What is port exposing versus port mapping?
Port exposing (--expose) makes a container’s port available only to other containers inside the

Docker network. Port mapping (-p hostPort:containerPort) connects a container's port to a port on the host machine, allowing external access.

2. Based on your experiments, what does port mapping achieve that just exposing a port does not?
Using docker ps, the exposed container shows PORTS as 80/tcp, meaning it's internal only. The mapped container shows 0.0.0.0:8080->80/tcp, which proves that port mapping allows access from the host, something exposing alone does not.

3. How does this affect external access to container services?
Port mapping forwards traffic from the host's port (8080) to the container's port (80), making the service accessible from the host machine. Exposing a port without mapping keeps the service unreachable from the host or any external system.



```
tul38268@cloudshell:~ (lab-2422)$ docker run -d --name nginx_mapped -p 8080:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
b1badc6e5066: Pull complete
a2da0c0f2353: Pull complete
e5d9bb0b85cc: Pull complete
14a859b5ba24: Pull complete
716cdf61af59: Pull complete
14e422fd20a0: Pull complete
c3741b707ce6: Pull complete
Digest: sha256:33e0bbc7ca9ecf108140af6288c7c9d1ecc77548cbfd3952fd8466a75edefe57
Status: Downloaded newer image for nginx:latest
b35944de95df64abc40a5d0420b75c9922397cadd48085951df96e74c2b7f0bc
```

docker run tells Docker to start a new container.

-d runs it in detached mode, which means it runs in the background instead of taking over your terminal.

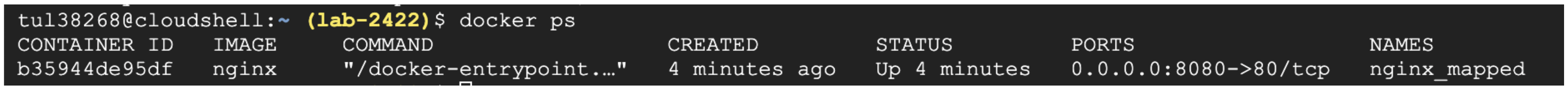
--name nginx_mapped gives the container a readable name (nginx_mapped) instead of you having to use the long container ID.

-p 8080:80 creates a port mapping:

The first number (8080) is the port on your computer (the host).

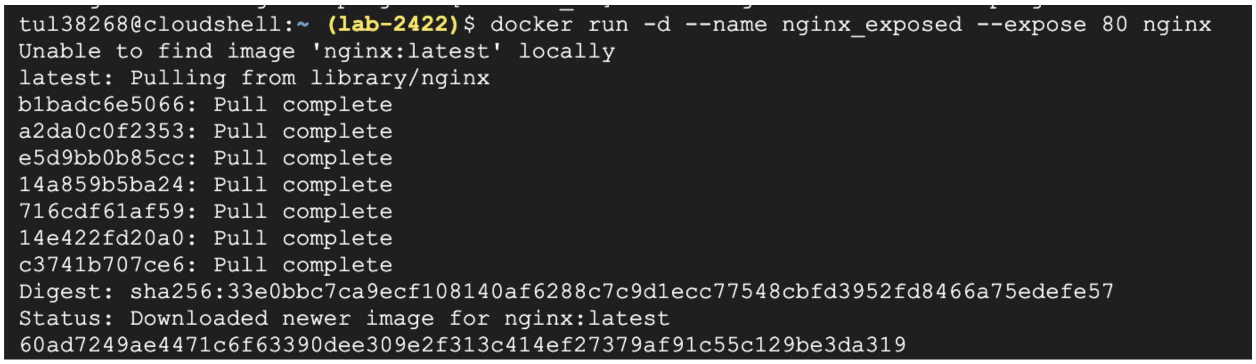
The second number (80) represents the port inside the container that NGINX is running on and handling requests.

The purpose of this command is to start an NGINX web server inside a container, run it in the background, give it the name nginx_mapped, and map the host machine's port 8080 to the container's port 80 so the server can be accessed externally through localhost:8080.



```
tul38268@cloudshell:~ (lab-2422)$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                  NAMES
b35944de95df   nginx     "/docker-entrypoint.…"   4 minutes ago   Up 4 minutes   0.0.0.0:8080->80/tcp   nginx_mapped
```

The command docker ps lists all active containers along with their IDs, names, status, and port mappings. For nginx_mapped, you should see something like 0.0.0.0:8080->80/tcp, which confirms that the container's port 80 is bound to the host's port 8080.



```
tul38268@cloudshell:~ (lab-2422)$ docker run -d --name nginx_exposed --expose 80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
b1badc6e5066: Pull complete
a2da0c0f2353: Pull complete
e5d9bb0b85cc: Pull complete
14a859b5ba24: Pull complete
716cdf61af59: Pull complete
14e422fd20a0: Pull complete
c3741b707ce6: Pull complete
Digest: sha256:33e0bbc7ca9ecf108140af6288c7c9d1ecc77548cbfd3952fd8466a75edefe57
Status: Downloaded newer image for nginx:latest
60ad7249ae4471c6f63390dee309e2f313c414ef27379af91c55c129be3da319
```

docker run → starts a new container.

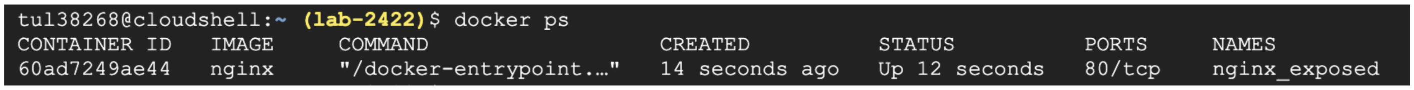
-d → runs it in the background (detached mode).

--name nginx_exposed → gives the container the name nginx_exposed.

--expose 80 → marks port 80 inside the container as available to Docker's internal network, but it does not connect it to your computer's ports. This means other containers in the same Docker network can talk to it, but you can't reach it directly from the host.

nginx → runs the official NGINX image.

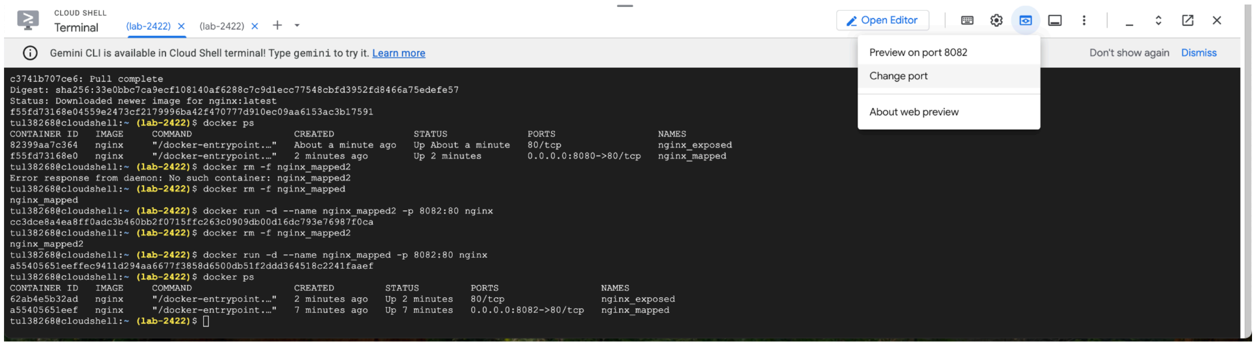
The purpose of this command is to start an NGINX web server inside a container, run it in the background, give it the name nginx_exposed, and expose port 80 so it can be accessed by other containers on the same Docker network, but not from the host machine or external systems.



```
tul38268@cloudshell:~ (lab-2422)$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
60ad7249ae44   nginx     "/docker-entrypoint.…"   14 seconds ago   Up 12 seconds   80/tcp    nginx_exposed
```

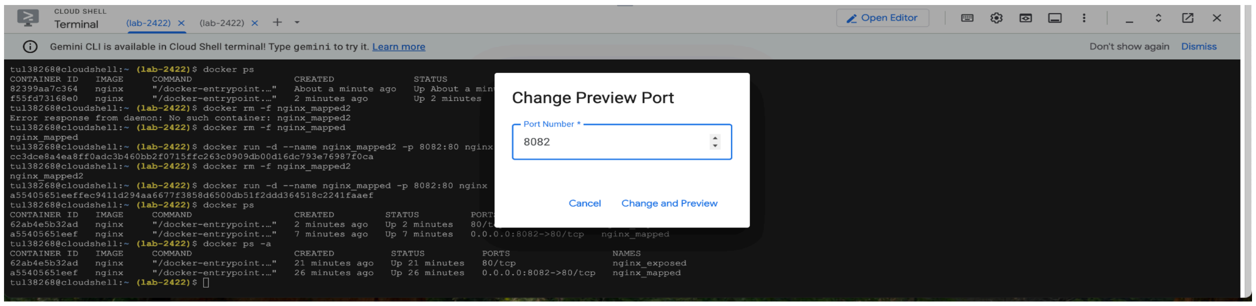
Notice: only 80/tcp is listed, meaning the port is visible inside Docker but not mapped to the host machine.

Proof that port mapping allows for external access:



Gemini CLI is available in Cloud Shell terminal! Type gemini to try it. Learn more

```
c3741b707ce6: Pull complete
Digest: sha256:33e0bbc7ca9ecf108140af6288c7c9d1ecc77548cbfd3952fd8466a75edefe57
Status: Downloaded newer image for nginx:latest
f55fd73168e04559e2473cf2179996ba42f470777d910ec09aa6153ac3b17591
tul38268@cloudshell:~ (lab-2422)$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS              PORTS                  NAMES
82399aa7c364   nginx     "/docker-entrypoint.…"   About a minute ago   Up About a minute   80/tcp                 nginx_exposed
f55fd73168e0   nginx     "/docker-entrypoint.…"   2 minutes ago        Up 2 minutes        0.0.0.0:8080->80/tcp   nginx_mapped
tul38268@cloudshell:~ (lab-2422)$ docker rm -f nginx_mapped2
Error response from daemon: No such container: nginx_mapped2
tul38268@cloudshell:~ (lab-2422)$ docker rm -f nginx_mapped
nginx_mapped
tul38268@cloudshell:~ (lab-2422)$ docker run -d --name nginx_mapped2 -p 8082:80 nginx
cc3dce8a4ea8ff0adc3b460bb2f0715ffc263c0909db00d16dc793e76987f0ca
tul38268@cloudshell:~ (lab-2422)$ docker rm -f nginx_mapped2
nginx_mapped2
tul38268@cloudshell:~ (lab-2422)$ docker run -d --name nginx_mapped -p 8082:80 nginx
a55405651eeffec9411d294aa6677f3858d6500db51f2ddd364518c2241faaef
tul38268@cloudshell:~ (lab-2422)$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                  NAMES
62ab4e5b32ad   nginx     "/docker-entrypoint.…"   2 minutes ago   Up 2 minutes   80/tcp                 nginx_exposed
a55405651eef   nginx     "/docker-entrypoint.…"   7 minutes ago   Up 7 minutes   0.0.0.0:8082->80/tcp   nginx_mapped
tul38268@cloudshell:~ (lab-2422)$ 
```

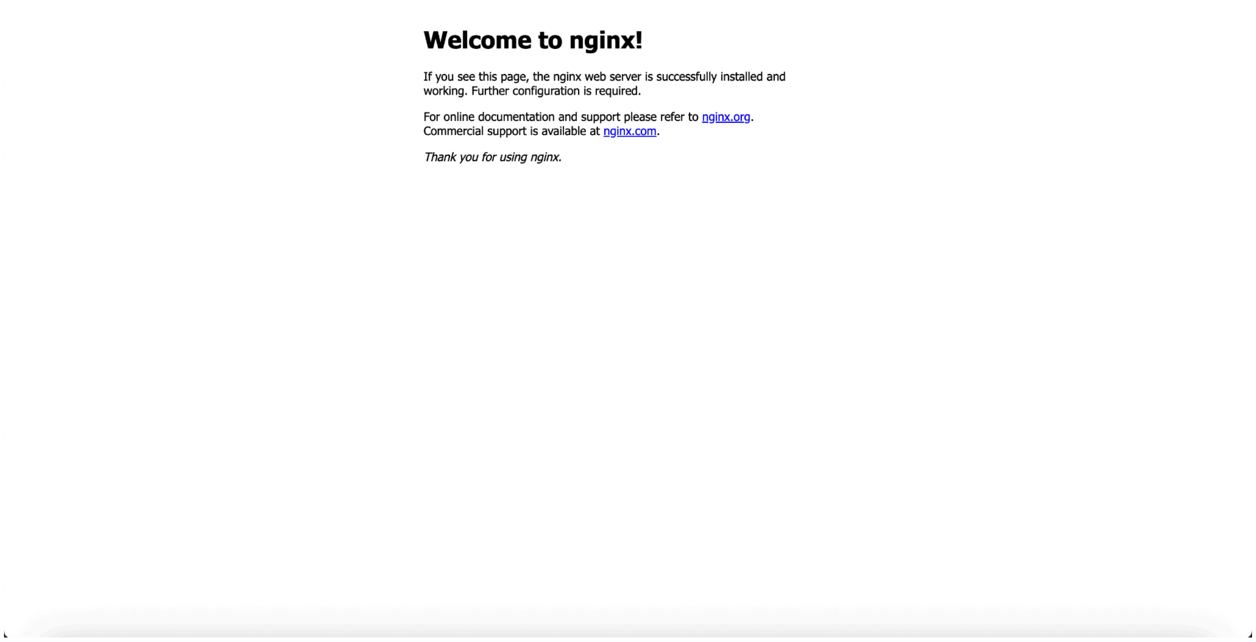


Gemini CLI is available in Cloud Shell terminal! Type gemini to try it. Learn more

```
tul38268@cloudshell:~ (lab-2422)$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS
82399aa7c364   nginx     "/docker-entrypoint.…"   About a minute ago   Up About a min
f55fd73168e0   nginx     "/docker-entrypoint.…"   2 minutes ago        Up 2 minutes
tul38268@cloudshell:~ (lab-2422)$ docker rm -f nginx_mapped2
Error response from daemon: No such container: nginx_mapped2
tul38268@cloudshell:~ (lab-2422)$ docker rm -f nginx_mapped
nginx_mapped
tul38268@cloudshell:~ (lab-2422)$ docker run -d --name nginx_mapped2 -p 8082:80 nginx
cc3dce8a4ea8ff0adc3b460bb2f0715ffc263c0909db00d16dc793e76987f0ca
tul38268@cloudshell:~ (lab-2422)$ docker rm -f nginx_mapped2
nginx_mapped2
tul38268@cloudshell:~ (lab-2422)$ docker run -d --name nginx_mapped -p 8082:80 nginx
a55405651eeffec9411d294aa6677f3858d6500db51f2ddd364518c2241faaef
tul38268@cloudshell:~ (lab-2422)$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORT
62ab4e5b32ad   nginx     "/docker-entrypoint.…"   2 minutes ago   Up 2 minutes   80/t
a55405651eef   nginx     "/docker-entrypoint.…"   7 minutes ago   Up 7 minutes   0.0.0.0:8082->80/tcp   nginx_mapped
tul38268@cloudshell:~ (lab-2422)$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                  NAMES
62ab4e5b32ad   nginx     "/docker-entrypoint.…"   21 minutes ago   Up 21 minutes   80/tcp                 nginx_exposed
a55405651eef   nginx     "/docker-entrypoint.…"   26 minutes ago   Up 26 minutes   0.0.0.0:8082->80/tcp   nginx_mapped
tul38268@cloudshell:~ (lab-2422)$ 
```

Change Preview Port

Port Number *: 8082



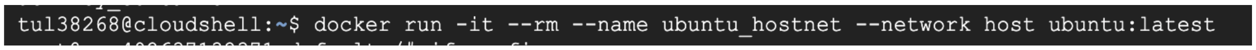
Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

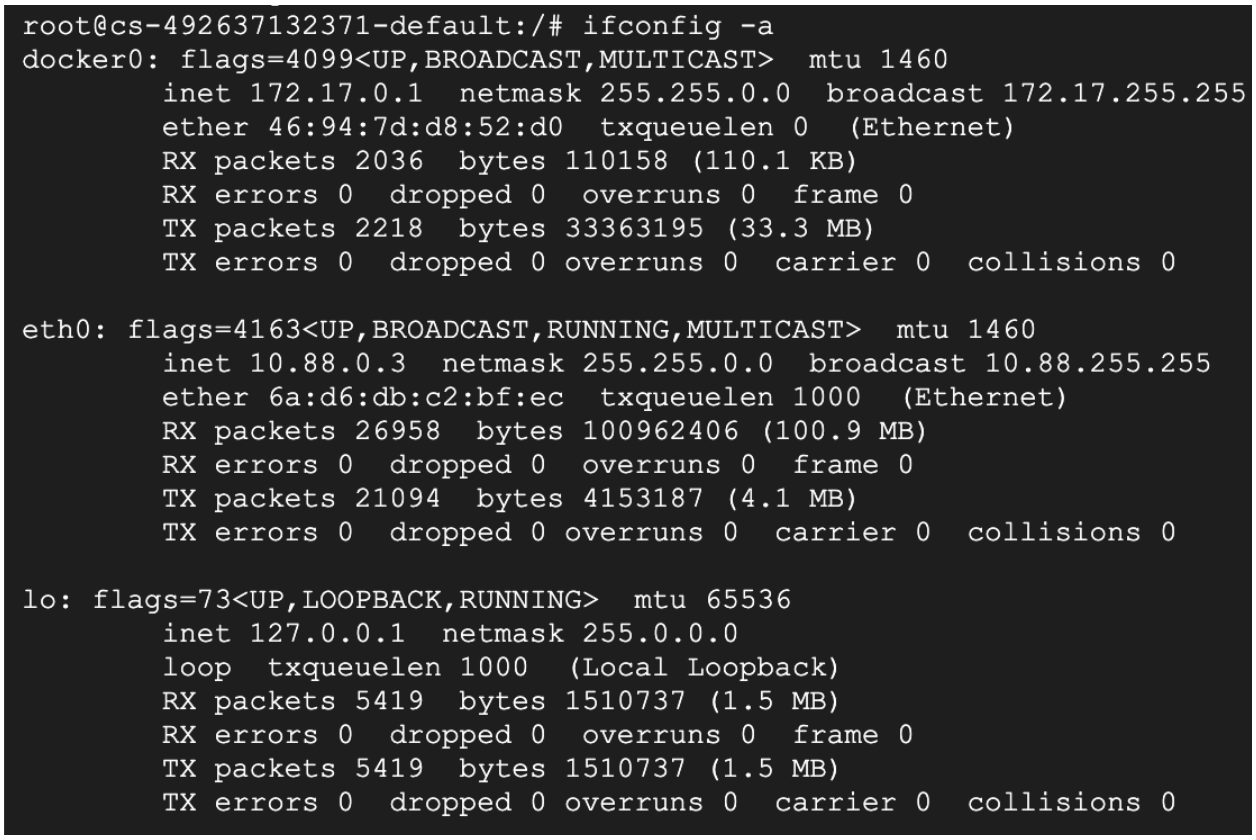
Thank you for using nginx.

Part 3



```
tul38268@cloudshell:~$ docker run -it --rm --name ubuntu_hostnet --network host ubuntu:latest
```

This command starts an interactive Ubuntu container that shares the host's network interfaces instead of being isolated.



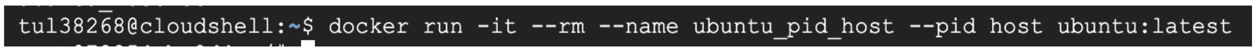
```
root@cs-492637132371-default:/# ifconfig -a
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1460
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 46:94:7d:d8:52:d0  txqueuelen 0  (Ethernet)
        RX packets 2036  bytes 110158 (110.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2218  bytes 33363195 (33.3 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1460
        inet 10.88.0.3  netmask 255.255.0.0  broadcast 10.88.255.255
        ether 6a:d6:db:c2:bf:ec  txqueuelen 1000  (Ethernet)
        RX packets 26958  bytes 100962406 (100.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 21094  bytes 4153187 (4.1 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 5419  bytes 1510737 (1.5 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5419  bytes 1510737 (1.5 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

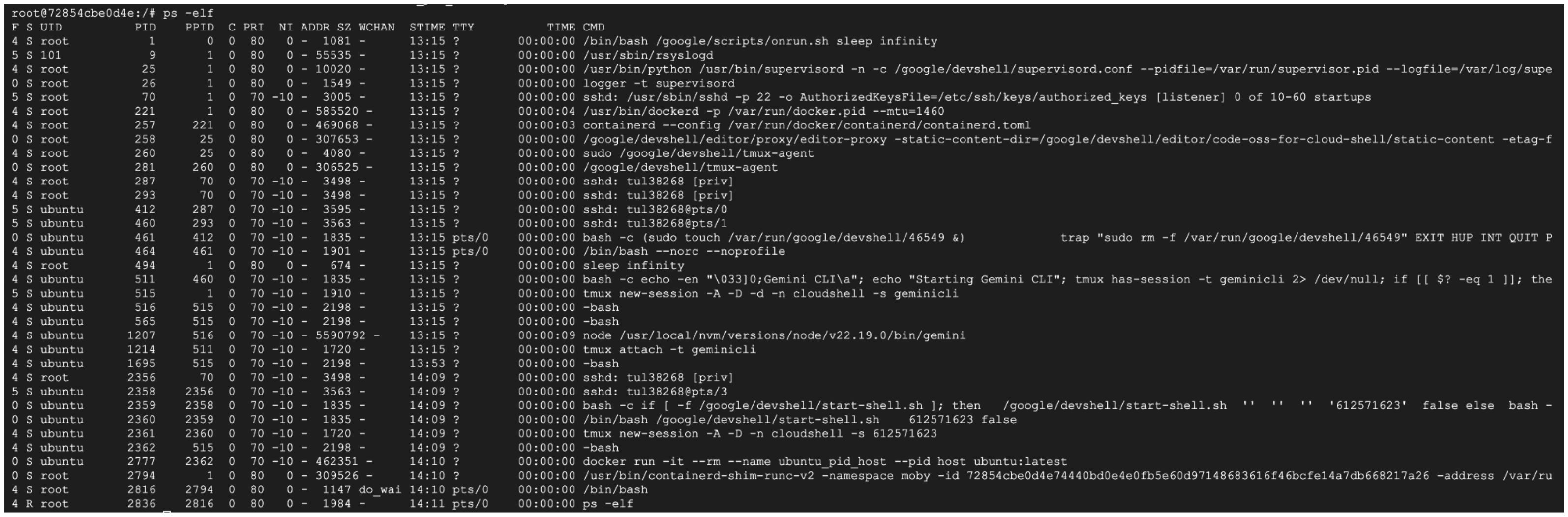
The command ifconfig -a inside the container started with docker run -it --rm --name ubuntu_hostnet --network host ubuntu:latest lists all network interfaces that the container sees, which are exactly the host's interfaces, showing that --network host makes the container use the host's network stack directly.

Problem: Sharing the host network with the container removes network isolation, which means the container can see, use, or interfere with all of the host's network interfaces and traffic, creating a serious security risk.



```
tul38268@cloudshell:~$ docker run -it --rm --name ubuntu_pid_host --pid host ubuntu:latest
```

The command starts an Ubuntu container that shares the host's process table, runs interactively, and removes itself on exit.



```
root@72854cbe0d4e:/# ps -elf
F S UID          PID    PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
4 S root           1       0  0  80   0 -  1081 -      13:15 ?        00:00:00 /bin/bash /google/scripts/onrun.sh sleep infinity
5 S 101            9       1  0  80   0 - 55535 -      13:15 ?        00:00:00 /usr/sbin/rsyslogd
4 S root          25       1  0  80   0 - 10020 -      13:15 ?        00:00:00 /usr/bin/python /usr/bin/supervisord -n -c /google/devshell/supervisord.conf --pidfile=/var/run/supervisor.pid --logfile=/var/log/supe
0 S root          26       1  0  80   0 -  1549 -      13:15 ?        00:00:00 logger -t supervisord
5 S root          70       1  0  70 -10 -  3005 -      13:15 ?        00:00:00 sshd: /usr/sbin/sshd -p 22 -o AuthorizedKeysFile=/etc/ssh/keys/authorized_keys [listener] 0 of 10-60 startups
4 S root         221       1  0  80   0 - 585520 -     13:15 ?        00:00:04 /usr/bin/dockerd -p /var/run/docker.pid --mtu=1460
4 S root         257     221  0  80   0 - 469068 -     13:15 ?        00:00:03 containerd --config /var/run/docker/containerd/containerd.toml
0 S root         258      25  0  80   0 - 307653 -     13:15 ?        00:00:00 /google/devshell/editor/proxy/editor-proxy -static-content-dir=/google/devshell/editor/code-oss-for-cloud-shell/static-content -etag-f
4 S root         260      25  0  80   0 -  4080 -      13:15 ?        00:00:00 sudo /google/devshell/tmux-agent
0 S root         281     260  0  80   0 - 306525 -     13:15 ?        00:00:00 /google/devshell/tmux-agent
4 S root         287      70  0  70 -10 -  3498 -      13:15 ?        00:00:00 sshd: tul38268 [priv]
4 S root         293      70  0  70 -10 -  3498 -      13:15 ?        00:00:00 sshd: tul38268 [priv]
5 S ubuntu       412     287  0  70 -10 -  3595 -      13:15 ?        00:00:00 sshd: tul38268@pts/0
5 S ubuntu       460     293  0  70 -10 -  3563 -      13:15 ?        00:00:00 sshd: tul38268@pts/1
0 S ubuntu       461     412  0  70 -10 -  1835 -      13:15 pts/0    00:00:00 bash -c (sudo touch /var/run/google/devshell/46549 &)            trap "sudo rm -f /var/run/google/devshell/46549" EXIT HUP INT QUIT P
4 S ubuntu       464     461  0  70 -10 -  1901 -      13:15 pts/0    00:00:00 /bin/bash --norc --noprofile
4 S root         494       1  0  80   0 -   674 -      13:15 ?        00:00:00 sleep infinity
4 S ubuntu       511     460  0  70 -10 -  1835 -      13:15 ?        00:00:00 bash -c echo -en "\033]0;Gemini CLI\a"; echo "Starting Gemini CLI"; tmux has-session -t geminicli 2> /dev/null; if [[ $? -eq 1 ]]; the
5 S ubuntu       515       1  0  70 -10 -  1910 -      13:15 ?        00:00:00 tmux new-session -A -D -d -n cloudshell -s geminicli
4 S ubuntu       516     515  0  70 -10 -  2198 -      13:15 ?        00:00:00 -bash
4 S ubuntu       565     515  0  70 -10 -  2198 -      13:15 ?        00:00:00 -bash
4 S ubuntu      1207     516  0  70 -10 - 5590792 -    13:15 ?        00:00:09 node /usr/local/nvm/versions/node/v22.19.0/bin/gemini
4 S ubuntu      1214     511  0  70 -10 -  1720 -      13:15 ?        00:00:00 tmux attach -t geminicli
4 S ubuntu      1695     515  0  70 -10 -  2198 -      13:53 ?        00:00:00 -bash
4 S root        2356      70  0  70 -10 -  3498 -      14:09 ?        00:00:00 sshd: tul38268 [priv]
5 S ubuntu      2358    2356  0  70 -10 -  3563 -      14:09 ?        00:00:00 sshd: tul38268@pts/3
0 S ubuntu      2359    2358  0  70 -10 -  1835 -      14:09 ?        00:00:00 bash -c if [ -f /google/devshell/start-shell.sh ]; then   /google/devshell/start-shell.sh  ''  ''  ''  '612571623'  false else  bash -
0 S ubuntu      2360    2359  0  70 -10 -  1835 -      14:09 ?        00:00:00 /bin/bash /google/devshell/start-shell.sh    612571623 false
4 S ubuntu      2361    2360  0  70 -10 -  1720 -      14:09 ?        00:00:00 tmux new-session -A -D -n cloudshell -s 612571623
4 S ubuntu      2362     515  0  70 -10 -  2198 -      14:09 ?        00:00:00 -bash
0 S ubuntu      2777    2362  0  70 -10 - 462351 -     14:10 ?        00:00:00 docker run -it --rm --name ubuntu_pid_host --pid host ubuntu:latest
0 S root        2794       1  0  80   0 - 309526 -     14:10 ?        00:00:00 /usr/bin/containerd-shim-runc-v2 -namespace moby -id 72854cbe0d4e74440bd0e4e0fb5e60d97148683616f46bcfe14a7db668217a26 -address /var/ru
4 S root        2816    2794  0  80   0 -  1147 do_wai 14:10 pts/0    00:00:00 /bin/bash
4 R root        2836    2816  0  80   0 -  1984 -      14:11 pts/0    00:00:00 ps -elf
```

Inside the container, ps -elf lists all processes in long format; it shows every process running on the host, not just container processes.

Problem: This also lowers security. A compromised container could spy on or even tamper with host processes, which breaks the isolation Docker is supposed to provide.

When the Ubuntu container uses --network host, it shares the host's network interfaces, so ifconfig -a shows the same IPs as the host. This is useful for debugging but removes network isolation, letting the container access or interfere with host traffic. With --pid host, the container shares the host's process table, so ps -elf shows all host processes. This allows process monitoring but breaks process isolation, letting the container view or manipulate host processes. Both options are powerful for testing but reduce security.