Task 6: Networking

- To start a ping container, we can use the following command:

```
ankit@sf-cpu-082:~/Documents/Assignment/Docker/docker-training/Task-6$ docker run --rm -it alpine ping google.com
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
f56be85fc22e: Pull complete
Digest: sha256:124c7d2707904eea7431fffe91522a01e5a861a624ee31d03372cc1d138a3126
Status: Downloaded newer image for alpine:latest
PING google.com (142.250.192.14): 56 data bytes
64 bytes from 142.250.192.14: seq=0 ttl=116 time=34.530 ms
64 bytes from 142.250.192.14: seq=1 ttl=116 time=34.711 ms
64 bytes from 142.250.192.14: seq=2 ttl=116 time=35.913 ms
64 bytes from 142.250.192.14: seq=3 ttl=116 time=35.481 ms
64 bytes from 142.250.192.14: seq=4 ttl=116 time=35.022 ms
64 bytes from 142.250.192.14: seq=5 ttl=116 time=36.421 ms
64 bytes from 142.250.192.14: seq=6 ttl=116 time=35.282 ms
64 bytes from 142.250.192.14: seq=7 ttl=116 time=35.097 ms
64 bytes from 142.250.192.14: seq=8 ttl=116 time=35.576 ms
64 bytes from 142.250.192.14: seq=9 ttl=116 time=34.666 ms
```

- To list all available network in docker

```
ankit@sf-cpu-082:~/Documents/Assignment/Docker/docker-training/Task-6$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
6b6b5e6c5e2a   bridge    bridge    local
76d93f3e0acc   host      host      local
04c489117c95   none      null      local
ankit@sf-cpu-082:~/Documents/Assignment/Docker/docker-training/Task-6$
```

- To inspect the bridge network, we can run:

```
ankit@sf-cpu-082:~/Documents/Assignment/Docker/docker-training/Task-6$ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "6b6b5e6c5e2a73b34c27a371def2b88ba242e641fbb68e48783c67b7581171b8",
        "Created": "2023-04-21T09:37:46.17803415+05:30",
        "Scope": "local",
        "Driver": "bridge",
        "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": "1500"
        },
        "Labels": {}
    }
]
ankit@sf-cpu-082:~/Documents/Assignment/Docker/docker-training/Task-6$
```

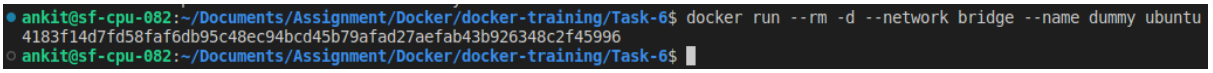
- To add the pinger container to the same network as the dummy container, we need to include the --network flag when running the pinger container.

- Assuming the network we created earlier is named "skynet", we can add the pinger container to the network with the following command:

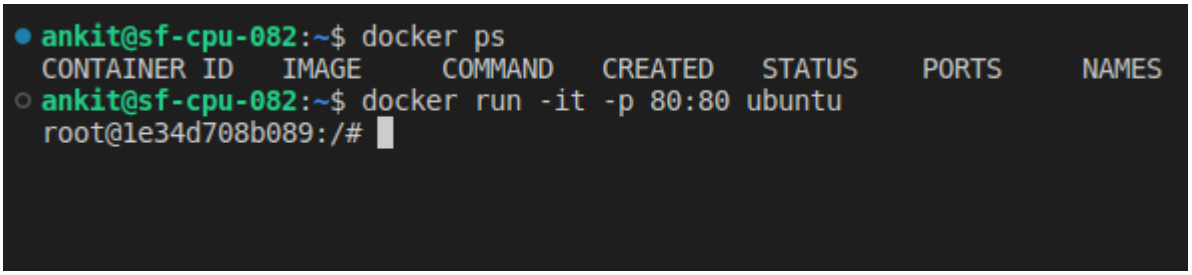
- To start the two postgres databases and add them to the same network, you can use the following commands:
    - docker run --rm -d --name widgetdb --network skynet -e POSTGRES_PASSWORD=mysecretpassword postgres

- This will start two postgres containers with the names widgetdb and gadgetdb, respectively, and add them to the skynet network. The -e flag sets the POSTGRES_PASSWORD environment variable, which is required for postgres to start up.

- Once the databases are running and on the same network, you can connect to one from the other using its container name as the host name. For example, to connect to the widgetdb database from the gadgetdb database, you would use the following command:



- Binding ports to the host



- Sometimes its useful to access an application running in a Docker container directly, as if it were running on your host machine.