Docker Network

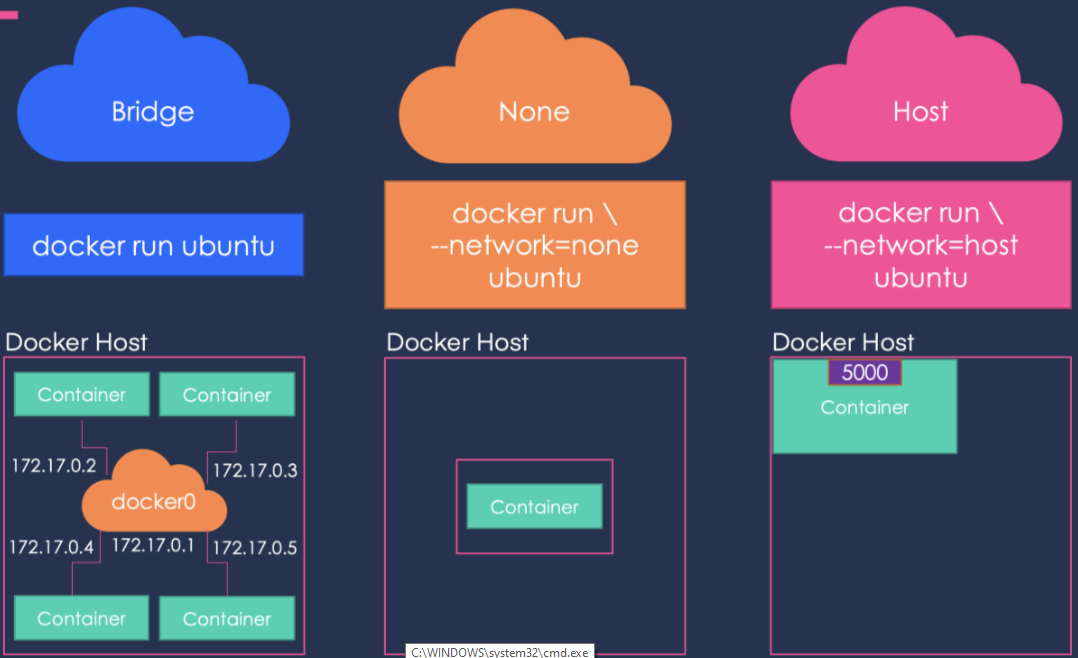
When we install docker it creates three types of networks automatically - Bridge, Host and None.

Bridge is default network and when we run any container it uses bridge network. To attach container to other network we can use –network flag in docker run command.

Bridge network: assigns IP to container in range of 172.17.x.x if container is within it. To access these containers from outside we need to explicitly map container post to ports of host machine.

Host network: this removes any isolation between container and host ex. if container has something hosted on port 80 then by using host machines port 80 we can directly access it without any explicit port mapping. Problem with this approach is same port can not be used on multiple containers.

None network: no network is attached to container; it is running in isolation and none of the port can be access form outside.

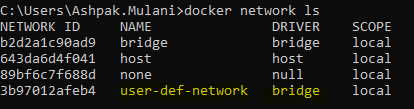


Bridge network (72.17.0.2) is by default attached to container but what if we want to attach our own network with own IP ? Yes we can create such network using following command

docker network create --driver bridge --subnet 182.18.0.0/16 user-def-network



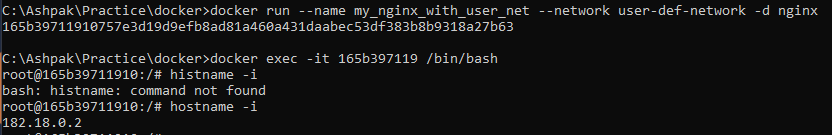
If we list network then we can see our newly added user-def-network



Now lest create a new container and attach it to our new network using –network tag

docker run --name my\_nginx\_with\_user\_net --network user-def-network -d nginx

if we check ip of container by getting into it then its ip is 182.18.0.2 in same range of new network



We can get details of specific network using ‘docker network inspect’ as shown below

docker network inspect user-def-network

this command returns all the details of specific network along with which containers are using it

[    {

        "Name": "user-def-network",

        "Id": "3b97012afeb4e74d5492784d53ad6e0727363fd4af5943f5a230630222b44169",

        "Created": "2022-09-17T09:53:29.4939926Z",

        "Scope": "local",

        "Driver": "bridge",

        "EnableIPv6": false,

        "IPAM": {

            "Driver": "default",

            "Options": {},

            "Config": [

                {

                    "Subnet": "182.18.0.0/16"

                }

            ]

        },

        "Internal": false,

        "Attachable": false,

        "Ingress": false,

        "ConfigFrom": {

            "Network": ""

        },

        "ConfigOnly": false,

        "Containers": {

            "165b39711910757e3d19d9efb8ad81a460a431daabec53df383b8b9318a27b63": {

                "Name": "my\_nginx\_with\_user\_net",

                "EndpointID": "ae81ed57d673acf20137e0e6990eecda4fa1ed293986b82105c31fd4b867fbdc",

                "MacAddress": "02:42:b6:12:00:02",

                "IPv4Address": "182.18.0.2/16",

                "IPv6Address": ""

            }

        },

        "Options": {},

        "Labels": {}

    }    ]

# **Sample project**

Setup a drupal application in one container with backend DB running in another container. Make sure both the containers are on same network so that drupal app can access backend DB.

**Step 1 :** Create a user define network

Name network drupal-network. If we don’t provide subnet then it automatically selects next subnet of default 172.17.0.0 In our example it has selected 172.18.0.0

docker network create drupal-network

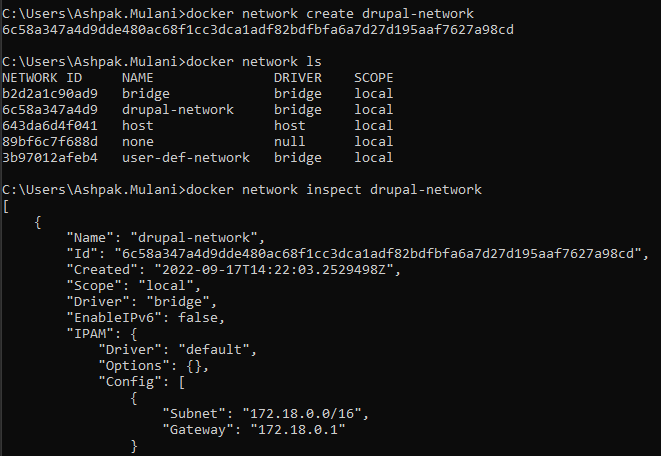
list networks to check if drupal-network creation is successful

docker network ls

inspect the newly created network

docker network inspect drupal-network

all commands combined looks like below



**Step 2 :** Create a persistent volume for database

Name persistent volume drupal-db-volume. IT will be used to store backend db for drupal application

docker volume create drupal-db-volume

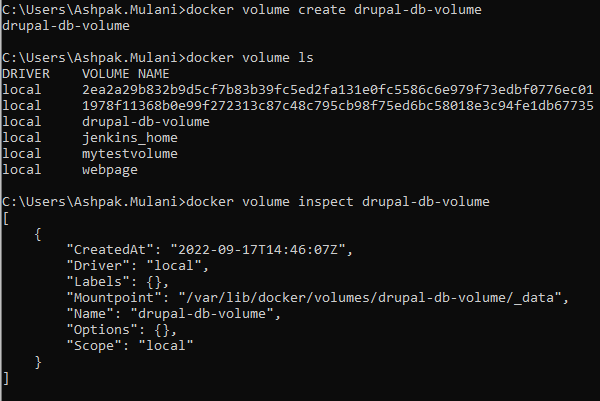
list all volumes to make sure our volume is created

docker volume ls

inspect the volume we created to get more details

docker volume inspect drupal-db-volume

all commands together look like below screenshot



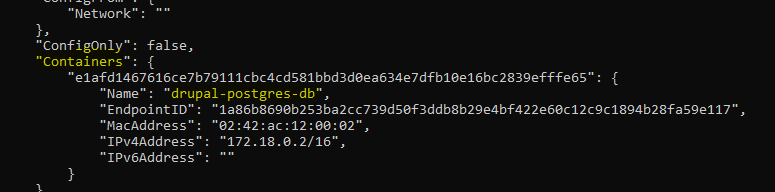
**Step 3 :** Start postgres DB container with attached network and volume from step 1 and step 2

Running docker container using postgres image in detached mode. While running we are specifying network and mount volume as well. As per documentation postgress container required to set few environment variables which are set using -e flag

docker run --name drupal-postgres-db --network drupal-network   
--mount src=drupal-db-volume,dst=/var/lib/postgresql/data   
-e POSTGRES\_DB=drupal -e POSTGRES\_USER=drupal -e OSTGRES\_PASSWORD=pw123 -d postgres:11.5

check if newly created container is attached to drupal-network

docker network inspect drupal-network



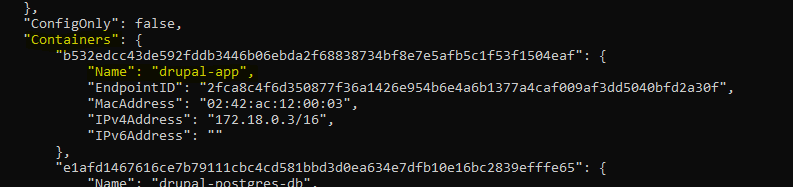
**Step 4 :** Start drupal application container

Start a container named "drupal" based on the "drupal" image with a tag of "8.7.7". Make sure it is connected to the "drupal" network. Also, publish port 80 on the host and map it to port 80 in the container.

docker run --name drupal-app --network drupal-network -p 80:80 -d drupal:8.7.7

check if newly created container is attached to drupal-network

docker network inspect drupal-network



**Step 5 :** configure drupal

Now drupal container is running and port 80 is mapped to local machine so we can access drupal app on <http://localhost> using host machine

While configuration drupal asks details of DB, where we can provide host name, db name, user name and password same as what weprovided dueing creation of DB container.

Please note here DB form other container is accessible by providing name of other container because each docker container can speak to other docker container (on **same user defined network**) by name.

After successful connection with DB drupal should start installing as shown on below screenshots.

