**Docker Bridge Network**

1. docker network ls
   1. Gives the network list on the host
2. docker network inspect bridge
   1. This gives a detailed information about the bridge network on our host. Take some time to examine it closely
3. ip link show docker0
   1. Native commands for linux also works
   2. The default “bridge” network, on all Linux-based Docker hosts, maps to an underlying Linux bridge in the kernel called “docker0”
4. docker network inspect bridge | grep bridge.name
   1. examine the output to know the relation between docker bridge and docker0
5. docker network create -d bridge localnet
   1. The new network is created and will appear in the output of any future docker network ls commands. If you are using Linux, you will also have a new Linux bridge created in the kernel.
6. brctl show
   1. use the Linux brctl tool to look at the Linux bridges currently on the system
   2. if brctl is not install, we can install it the command suggested in the terminal.
7. docker container run -d --name c1 --network localnet alpine sleep 1d
   1. a new alpine container is created and attached to the network created above.
8. docker network inspect localnet --format '{{json .Containers}}'
   1. check to see if the container is on the new network
9. brctl show
   1. check the value in the interfaces column.
   2. The value is c1’s interface attached to the new bridge.
10. If we add another new container to the same network, it should be able to ping the “c1” container by name. This is because all new containers are automatically registered with the embedded Docker DNS service, enabling them to resolve the names of all other containers on the same network.
11. The default bridge network on Linux does not support name resolution via the Docker DNS service. All other user-defined bridge networks do. The following demo will work because the container is on the user-defined localnet network.
12. docker container run -it --name c2 --network localnet alpine sh
    1. A new container c2 is created on the new bridge
13. ping c1
    1. from within c2 ping c1 by name.
    2. It works because the c2 container is running a local DNS resolver that forwards requests to an internal Docker DNS server. This DNS server maintains mappings for all containers started with the --name or --net-alias flag.
14. Port mappings let you map a container to a port on the Docker host. Any traffic hitting the Docker host on the configured port will be directed to the container
15. docker container run -d --name web --network localnet --publish 5000:80 nginx
    1. Run a new web server container and map port 80 on the container to port 5000 on the Docker host.
16. docker port web
    1. Verify the port mapping.
17. localhost:5000
    1. check this on docker host
    2. you can also use the base windows machine for this. Replace the IP of docker host instead of localhost
18. Mapping ports like this works, but it doesn’t scale. only a single container can bind to any port on the host. This means no other containers on that host will be able to bind to port 5000. This is one of the reason’s that single-host bridge networks are only useful for local development and very small applications.
19. Overlay networks are multi-host.
20. The ability to connect containerized apps to external systems and physical networks is vital. A common example is a partially containerized app — the containerized parts need a way to communicate with the non-containerized parts still running on existing physical networks and VLANs.
21. The built-in MACVLAN driver (transparent on Windows) was created with this in mind. It makes containers first-class citizens on the existing physical networks by giving each one its own MAC address and IP addresses.