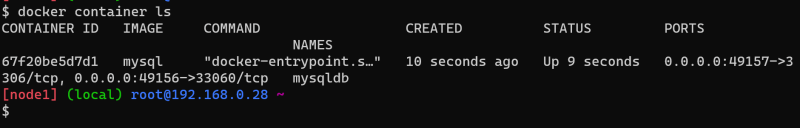
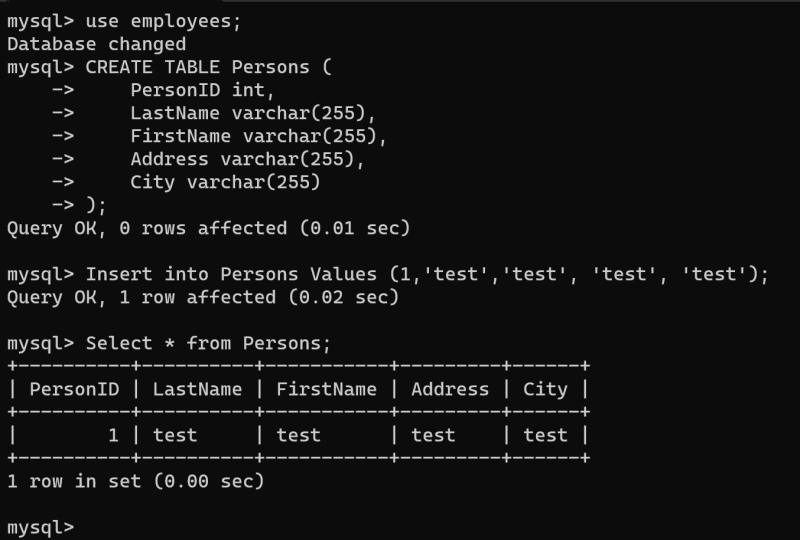
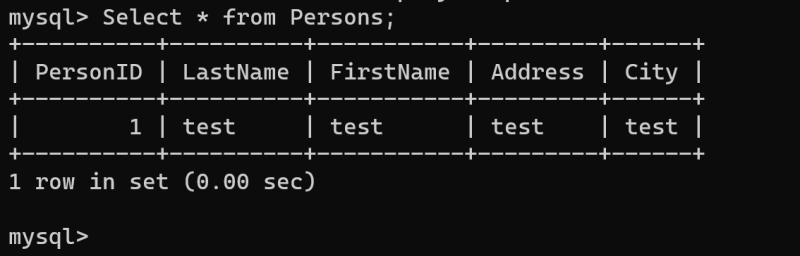
**Persisting Data using Volumes**

* Lets create an explicit volume for mysqldb
* Lets use volume type to mount the mysqldb
* Lets mount a volume using -v <https://docs.docker.com/storage/volumes/#start-a-container-with-a-volume> for official docs
* Create a mysql container

docker container run -d --name mysqldb -v mysqldb:/var/lib/mysql -P -e MYSQL\_ROOT\_PASSWORD=rootroot -e MYSQL\_DATABASE=employees -e MYSQL\_USER=qtdevops -e MYSQL\_PASSWORD=rootroot mysql

  
\* Lets create some data  
  
\* now delete the container  
\* Now create a new container using mount

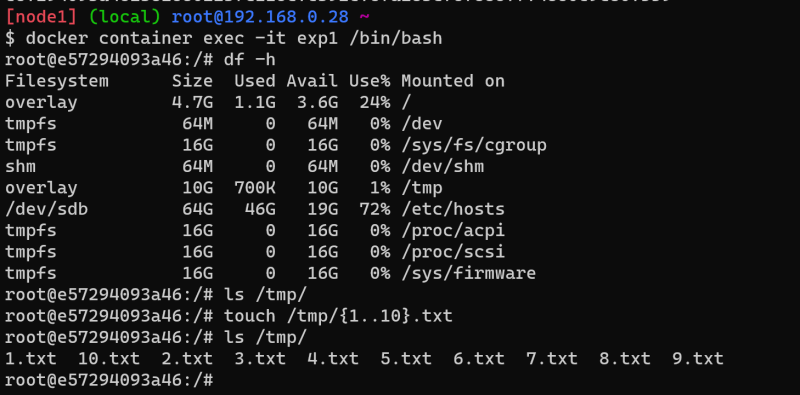
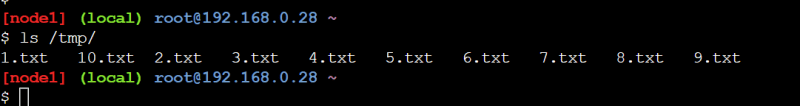
docker container run -d --name mysqldb --mount "source=mysqldb,target=/var/lib/mysql,type=volume" -P -e MYSQL\_ROOT\_PASSWORD=rootroot -e MYSQL\_DATABASE=employees -e MYSQL\_USER=qtdevops -e MYSQL\_PASSWORD=rootroot mysql

  
\* As we can see the data is persisted and is attached to new container.

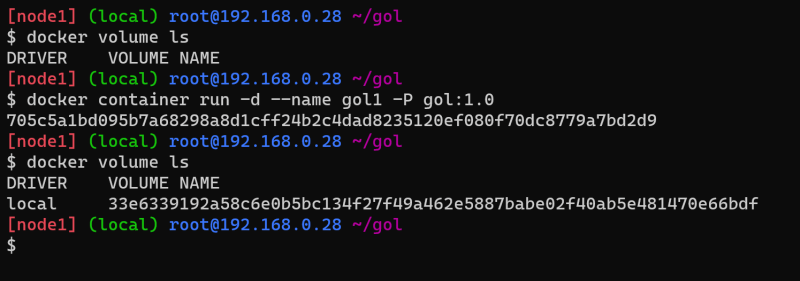
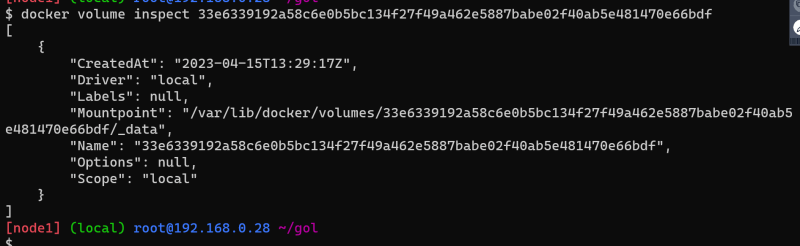
* Lets use bindmount to mount /tmp on docker host to the container /tmp

docker container run -d --name exp1 -v /tmp:/tmp ubuntu:22.04 sleep 1d

docker container run -d --name exp1 --mount "source=/tmp,target=/tmp,type=bind" ubuntu:22.04 sleep 1d

* Login into container and create some data  
  
* check the content in /tmp of docker host  
  

**Creating volume as part of Dockerfile**

* Gameoflife:
* <https://github.com/asquarezone/DockerZone/commit/f58812733781b7ebed7b2a8d0b0584fe0338c4e6> for the changeset with volume instruction for gameoflife container  
    
  

**Shell file to clean everything**

* Create a shell file with following content

#!/bin/bash

docker container rm -f $(docker container ls -a -q)

docker volume prune

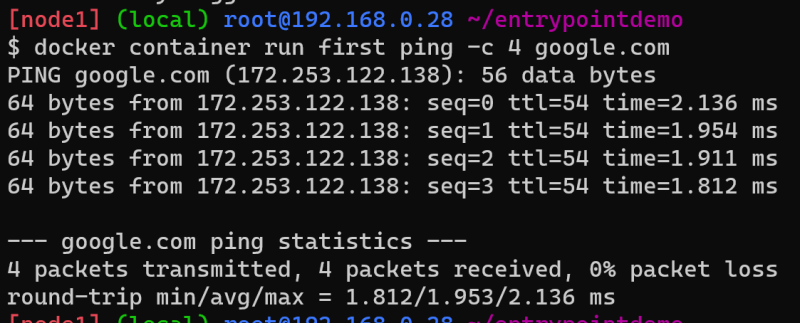
docker image rm -f $(docker image ls -q)

**Entrypoint and CMD**

* Lets create two docker images
* First

FROM alpine

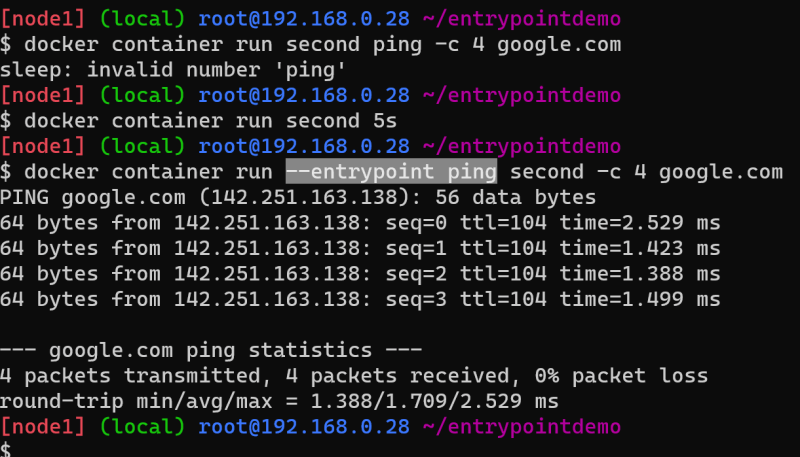
CMD ["sleep", "1d"]

* Create a container with docker container run first ping -c 4 google.com  
  
* Second

FROM alpine

ENTRYPOINT ["sleep"]

CMD ["1d"]



**Experiment for docker networking**

* Create a alpine container with the following names

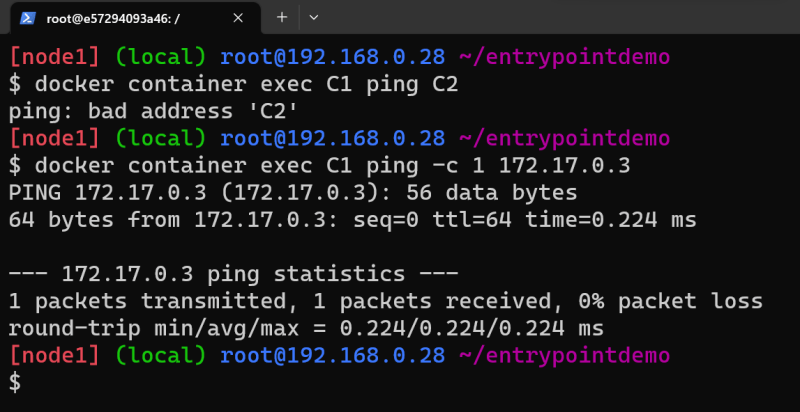
docker container run -d --name C1 alpine sleep 1d

docker container run -d --name C2 alpine sleep 1d

* Now run in C1 ping C2 (docker container exec C1 ping C2)
* Findout ip addresses of C1 container and C2 container by executing

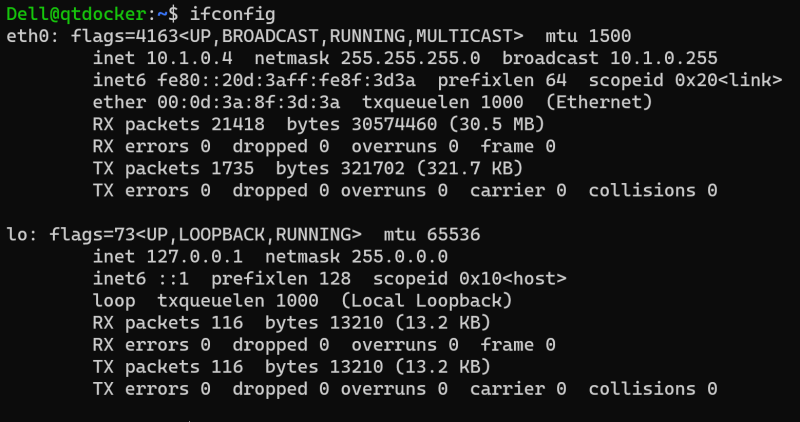
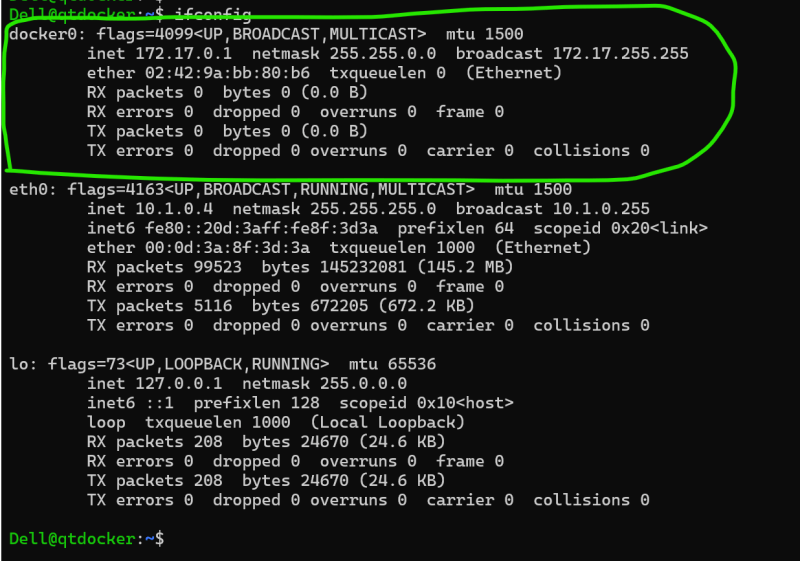
docker container inspect C1

docker container inspect C2

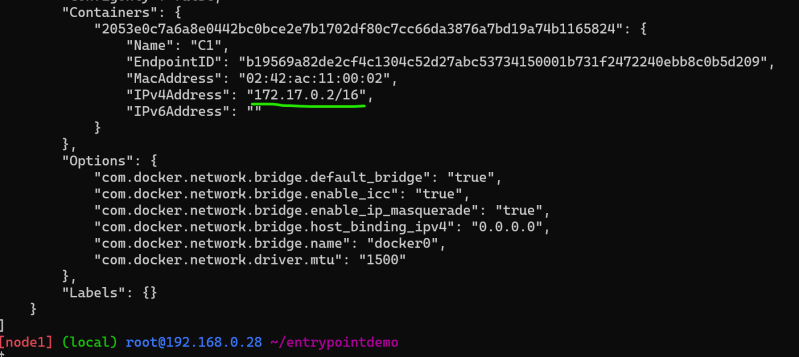
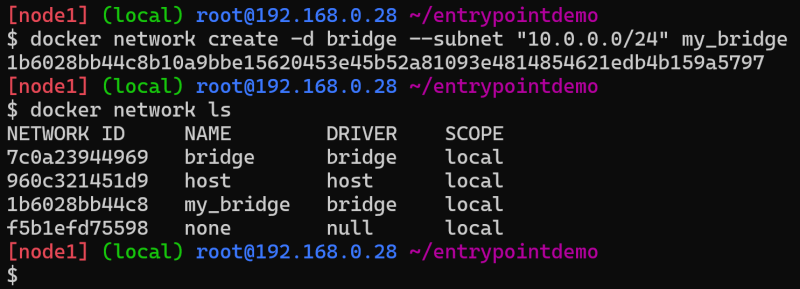
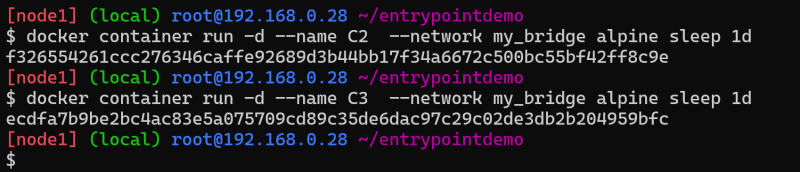
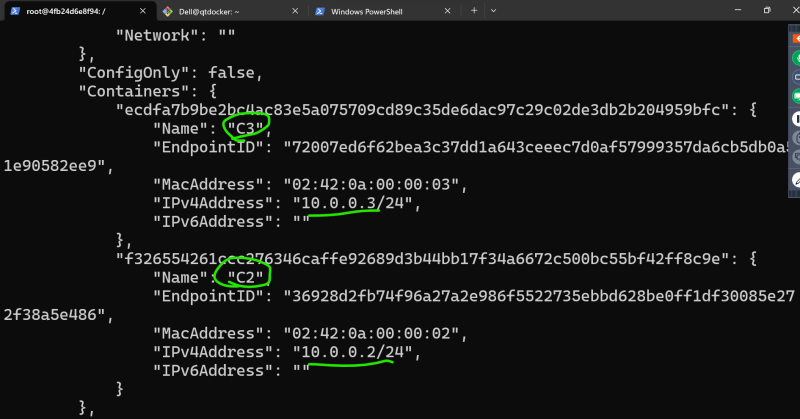
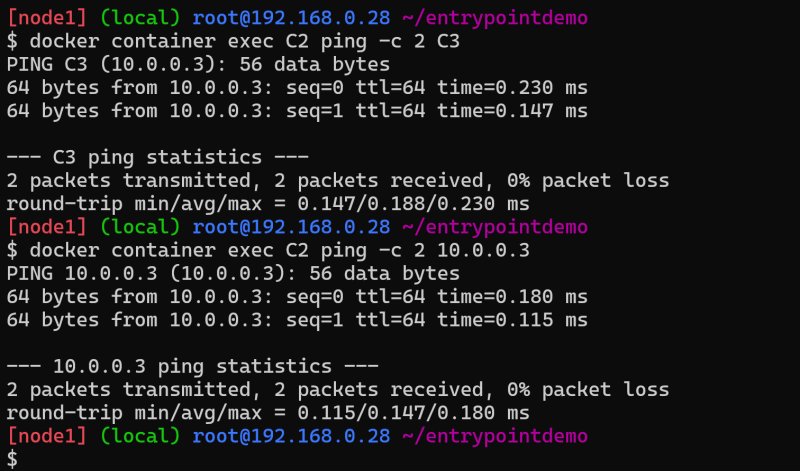
* Now login into C1 and ping C2 by using its ipaddress
* Observation Results
  + ping by name is not working
  + ping by ip is working  
    
* Create an ubunutu linux vm
* install net-tools

sudo apt update && sudo apt install net-tools -y

ifconfig

  
\* Now install docker and check network interfaces again ifconfig  
  
\* A docker0 network interface is added.

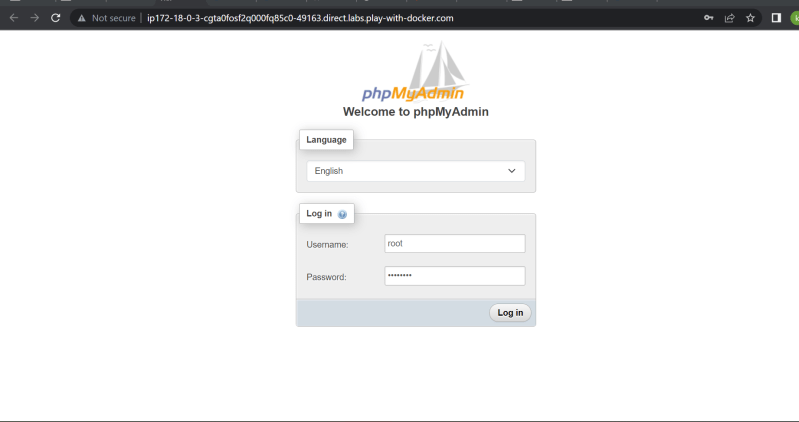
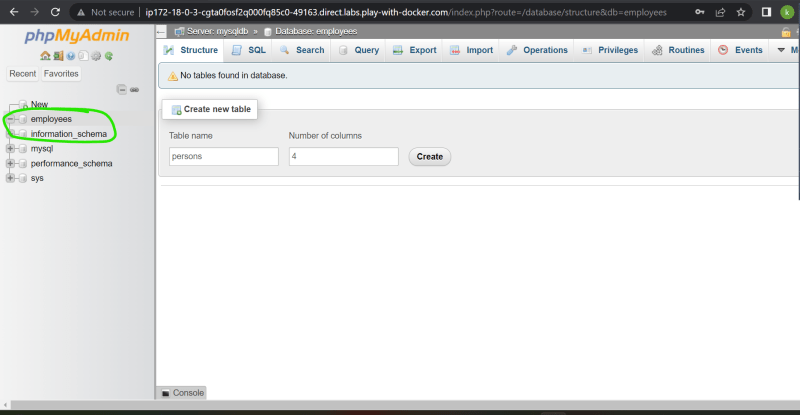
**Docker Networks**

* <https://directdevops.blog/2019/10/05/docker-networking-series-i/> for the article on docker networking
* Docker has multiple network driver implementations
  + bridge
  + host
  + macvlan
  + overlay
* Bridge:
  + Default bridge will not have dns enabled (this is why in the above experiment C1 was not able to ping C2 by name)
* Create a container C1 in default network docker container run -d --name C1 alpine sleep 1d
* inspect default bridge network docker network inspect bridge  
  
* Lets create a new bridge network  
  
* Now create two contianers C2 and C3 in my\_bridge network  
  
* Inspect my\_bridge network  
  
* Lets try ping from c2 to c3 by name  
  
* Lets create a mysql container in my\_bridge network

docker container run -d --name mysqldb -v mysqldb:/var/lib/mysql -P -e MYSQL\_ROOT\_PASSWORD=rootroot -e MYSQL\_DATABASE=employees -e MYSQL\_USER=qtdevops -e MYSQL\_PASSWORD=rootroot --network my\_bridge mysql

* Lets run phpmyadmin

docker container run --name phpmyadmin --network my\_bridge -d -e PMA\_HOST=mysqldb -P phpmyadmin

  
  
\* Connect container C1 to my\_bridge network

docker container exec C1 ip addr

docker network connect my\_bridge C1

docker container exec C1 ip addr

docker network disconnect bridge C1

docker container exec C1 ip addr