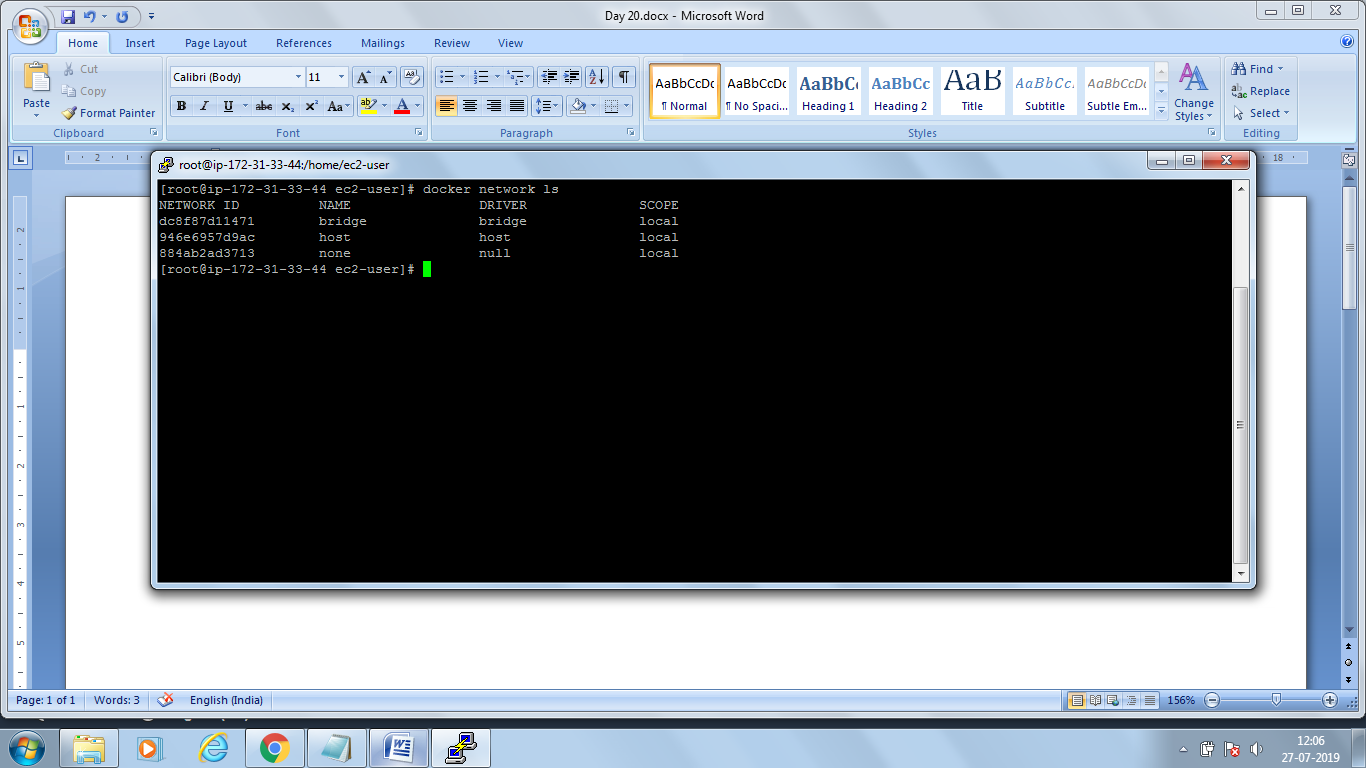
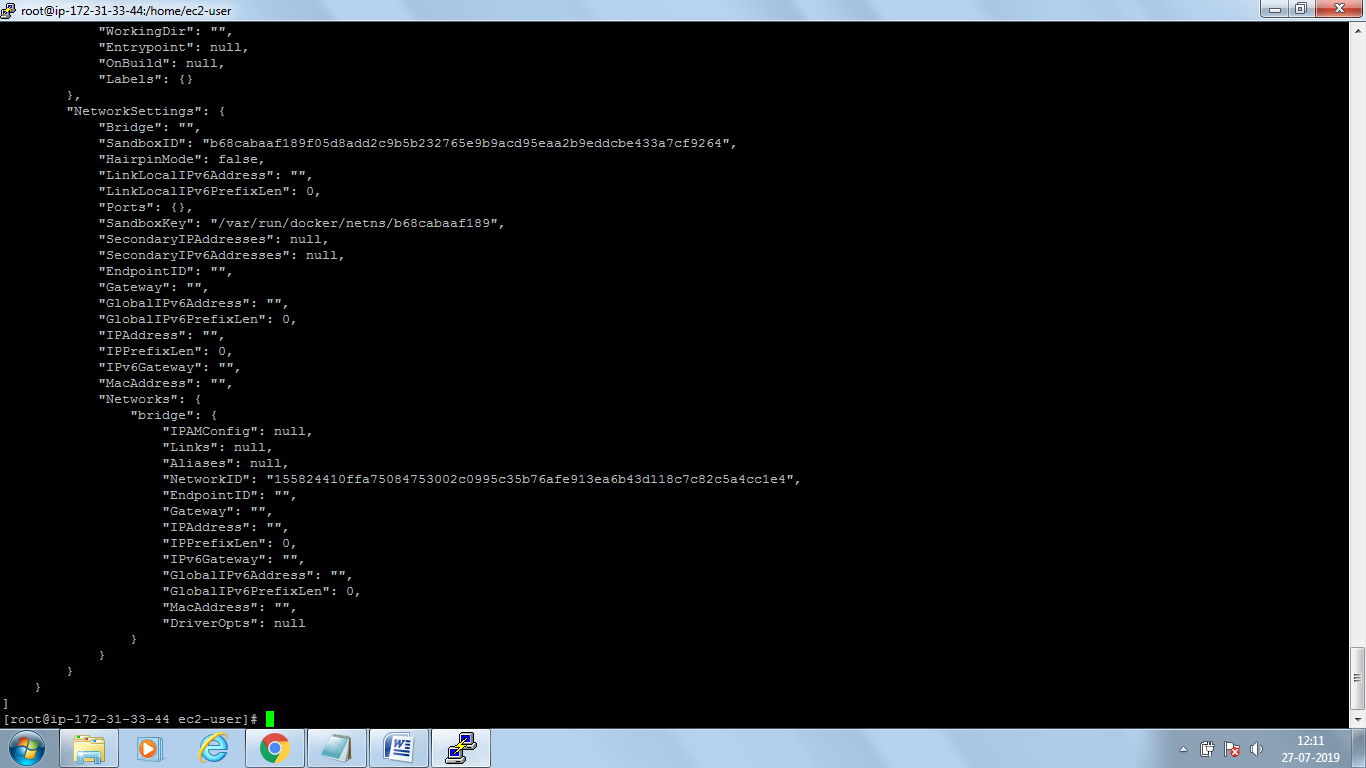
Docker network ls

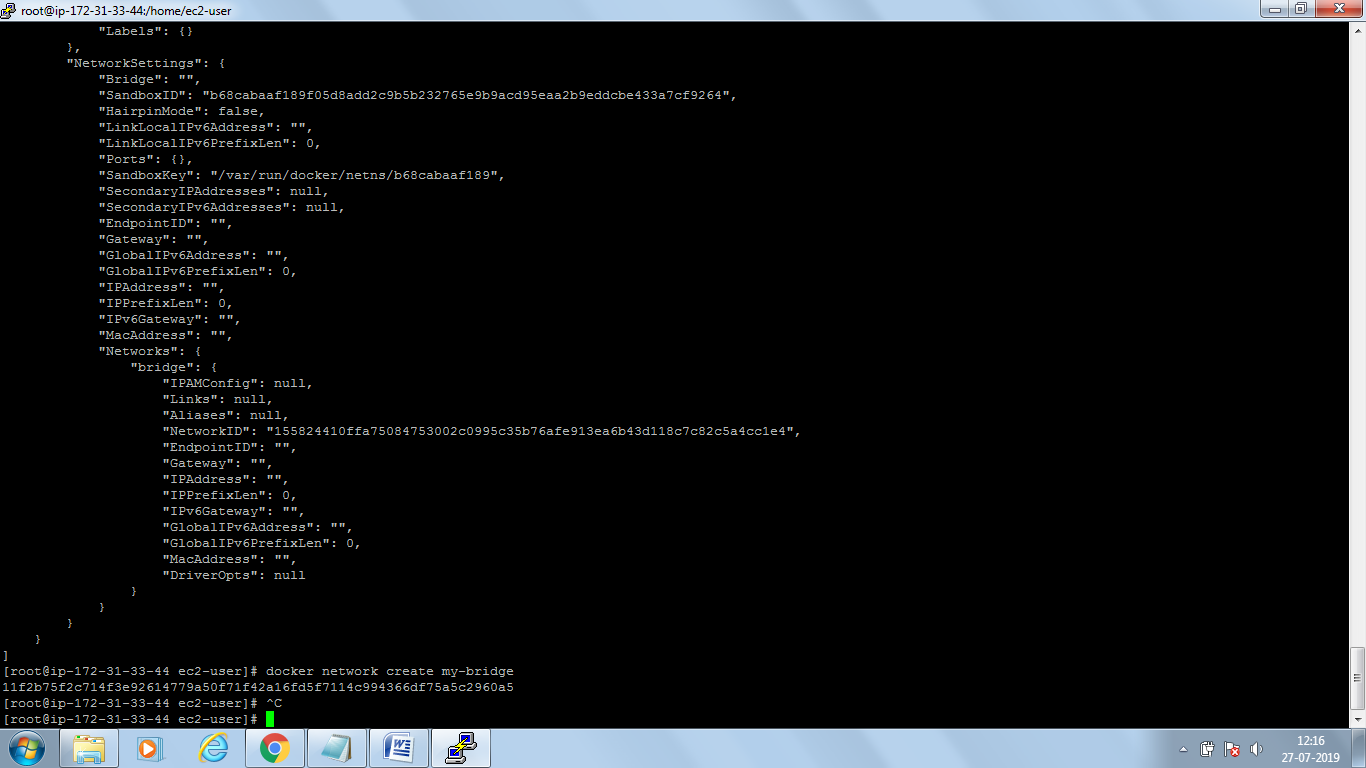


Default network

1. Bridge-container to container communication
2. Host:-container to host vm
3. None:-no connectivity (to get isolated environment)



docker network create my-bridge



Created few container and mapped to different port, noted down IP address

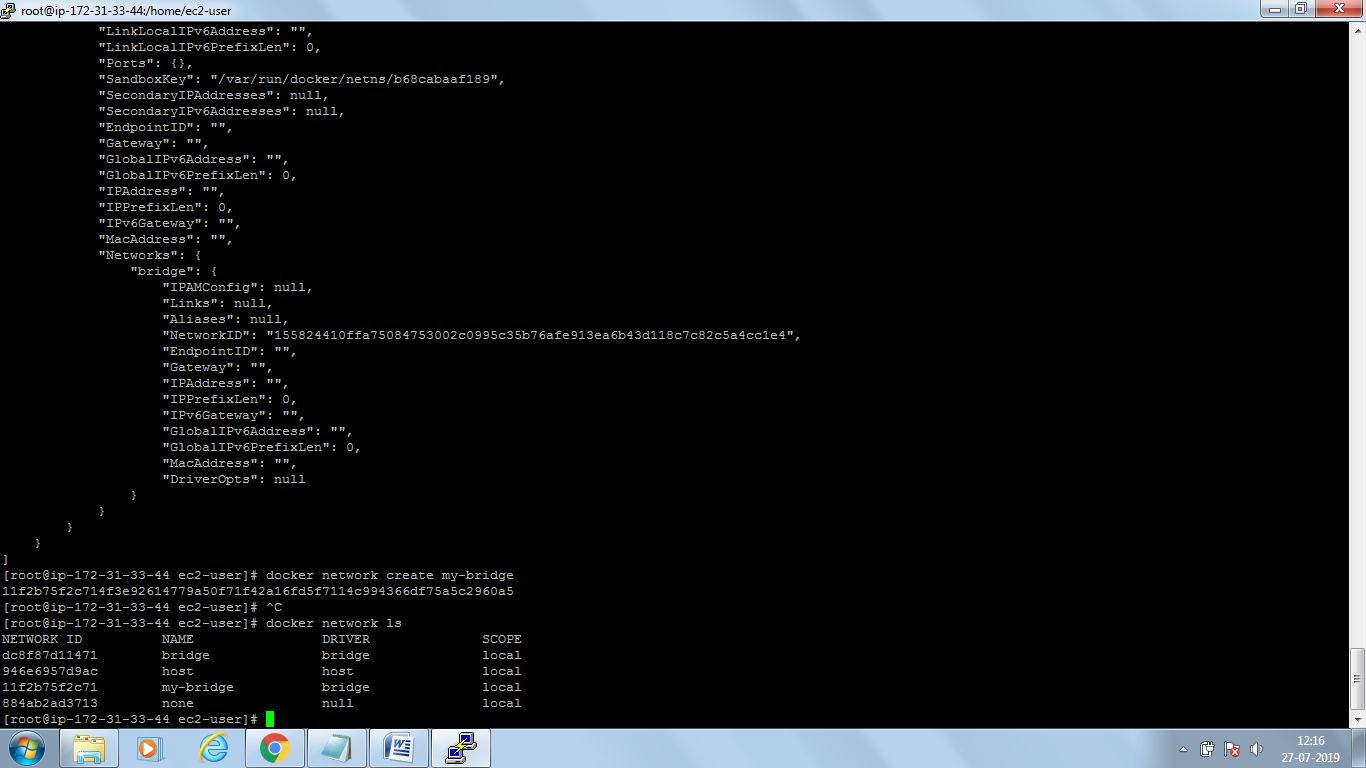
docker run –id –p 9001:80 jenkins

bfec67f92e69 tomcat --- 10001 "IPAddress": "172.17.0.3"

48415aa86027 nginx --- 10002 "IPAddress": "172.17.0.2"

apt-get install iputils-ping (not working)

docker inspect (you will find container will get on same default bridge)



175 docker run -itd -p 10001:80 nginx

176 docker run -itd -p 10002:80 tomcat

177 docker ps -a

178 docker inspect bfec67f92e69

179 docker inspect 48415aa86027

180 apt-get install iputils-ping

181 yum install inputils-ping

183 apt-get update

184 uname -a

187 docker run -itd -p 1000:80 centos

189 yum install iputils-ping

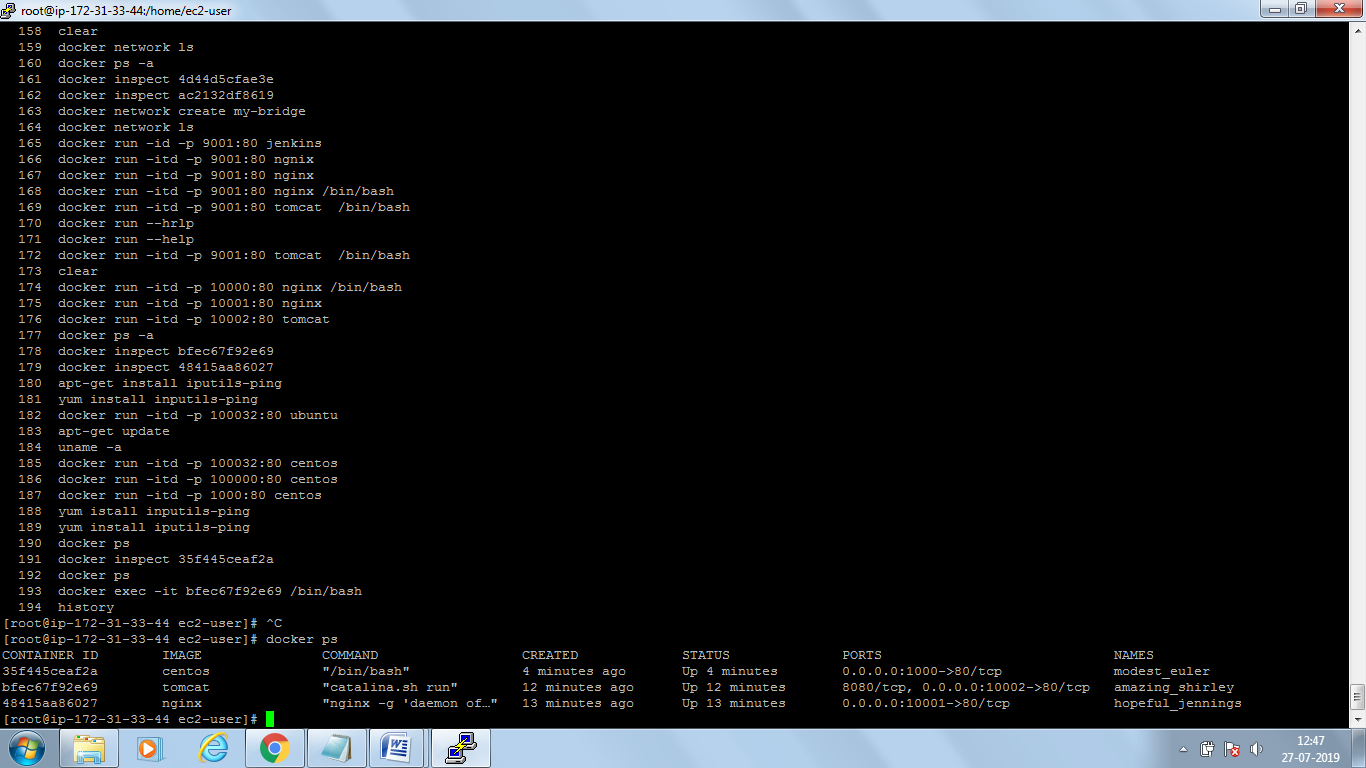
190 docker ps

191 docker inspect 35f445ceaf2a

192 docker ps

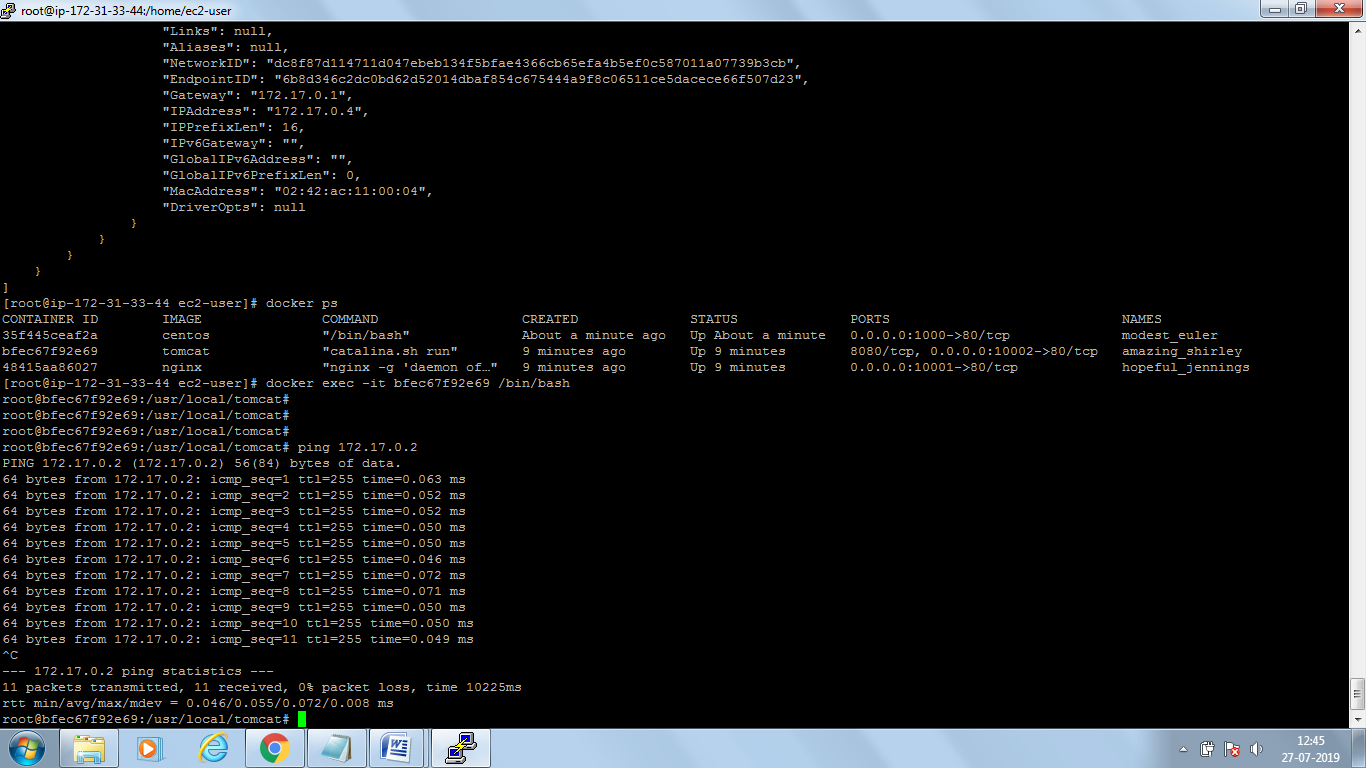
Now get inside one of container

193 docker exec -it bfec67f92e69 /bin/bash

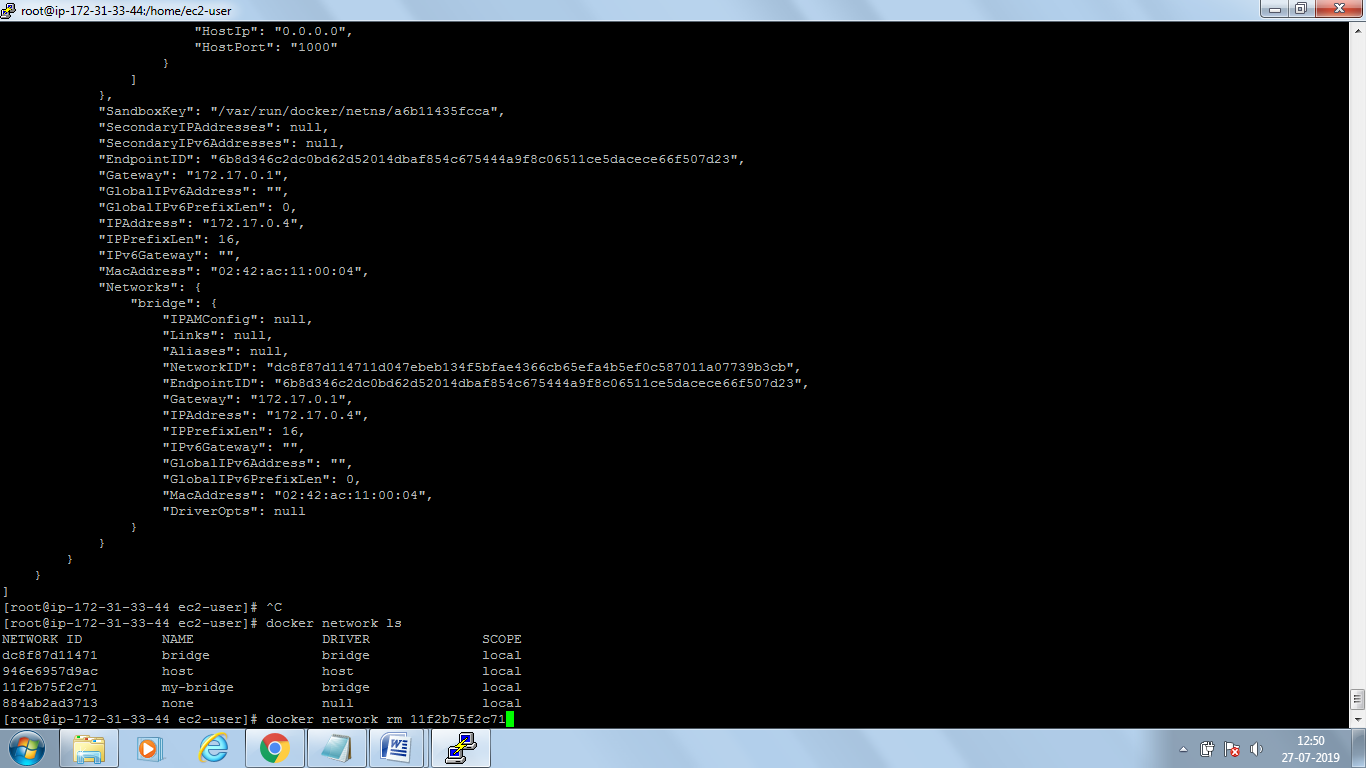


And ping the different container IP,ping should be done from that conatiner

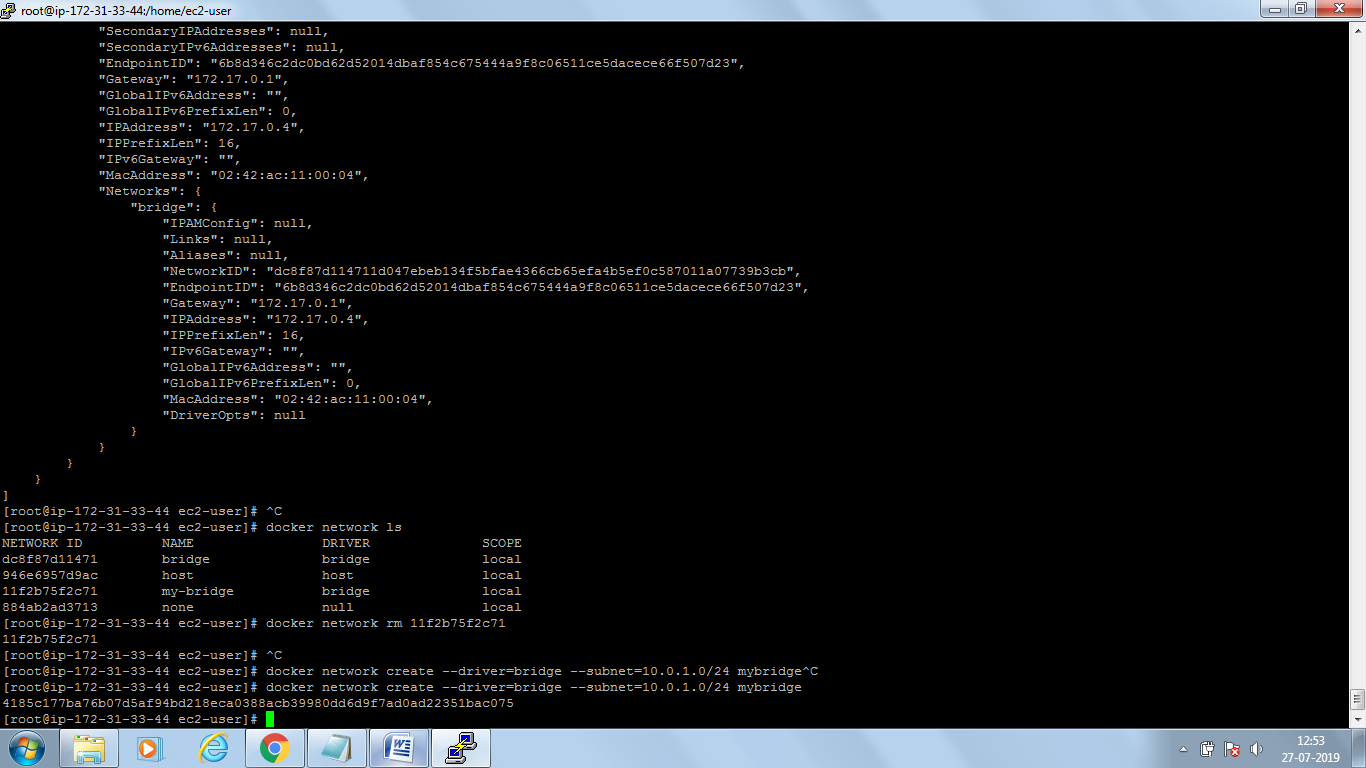
Ping 172.17.0.2

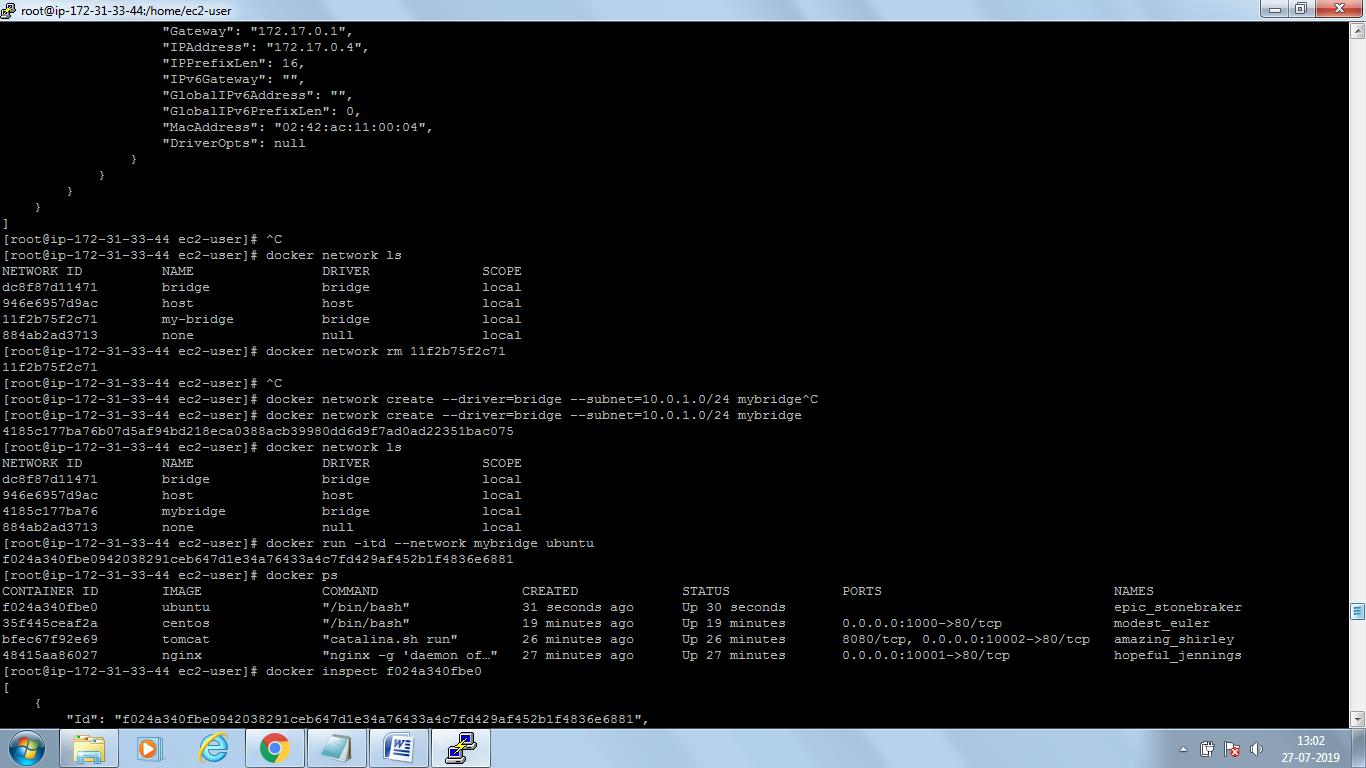


docker network rm 11f2b75f2c71



docker network create --driver=bridge --subnet=10.0.1.0/24 mybridge





U can see its created on different ip i.e.subnet

"IPAddress": "10.0.1.2",

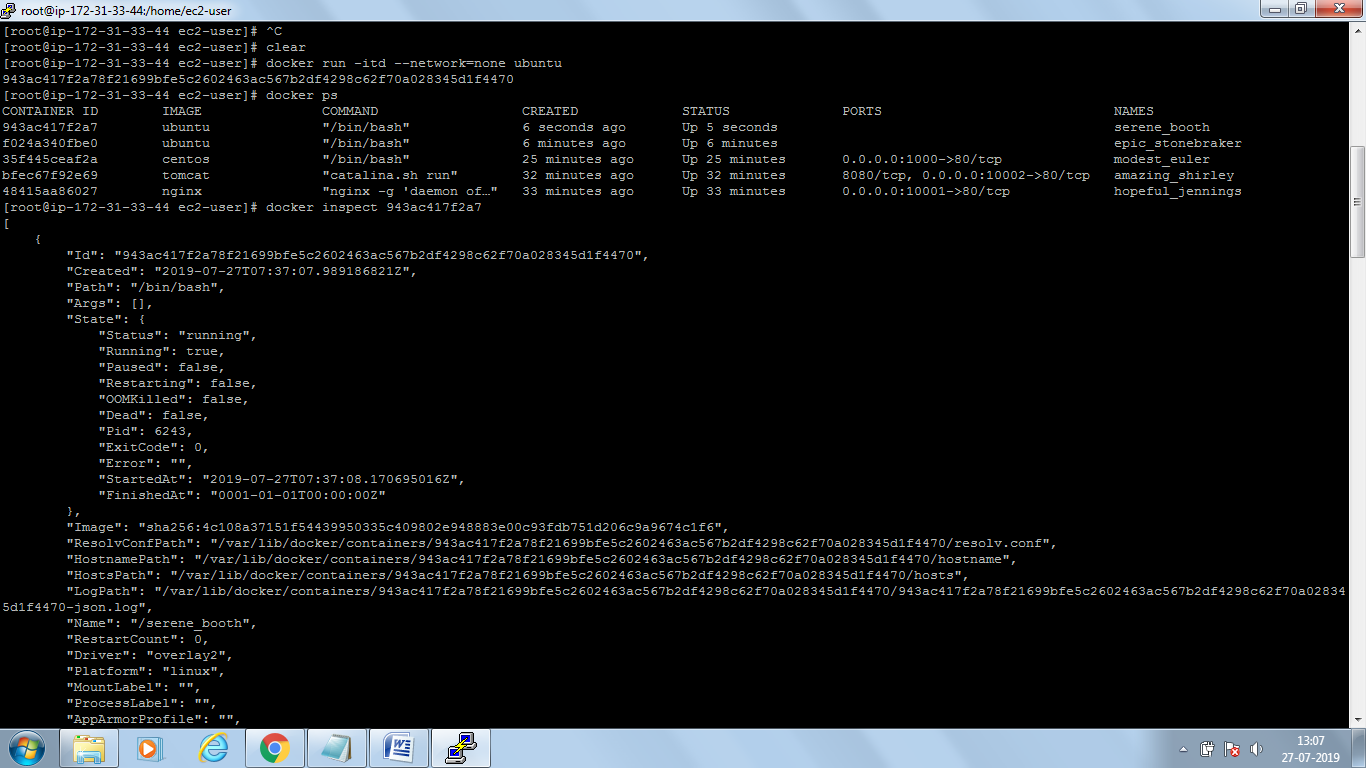
there is no communication between default bridge and n custom bridge (subnet)

none – lab –to create isolated container—mainly used for business critical data

205 docker run -itd --network=none ubuntu

206 docker ps

207 docker inspect 943ac417f2a7



Business critical data—isolated environment – security backup(none network)

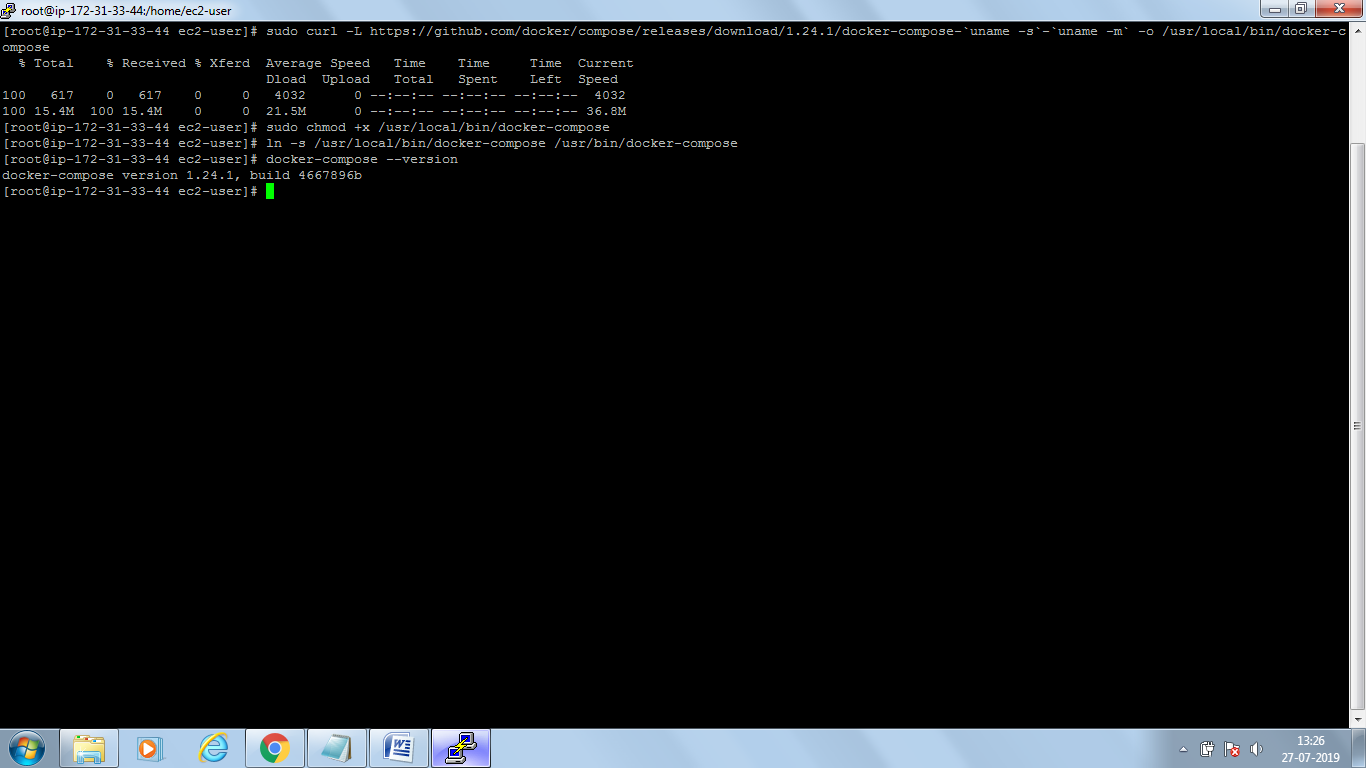
============== Docker compose ===============installation step==========

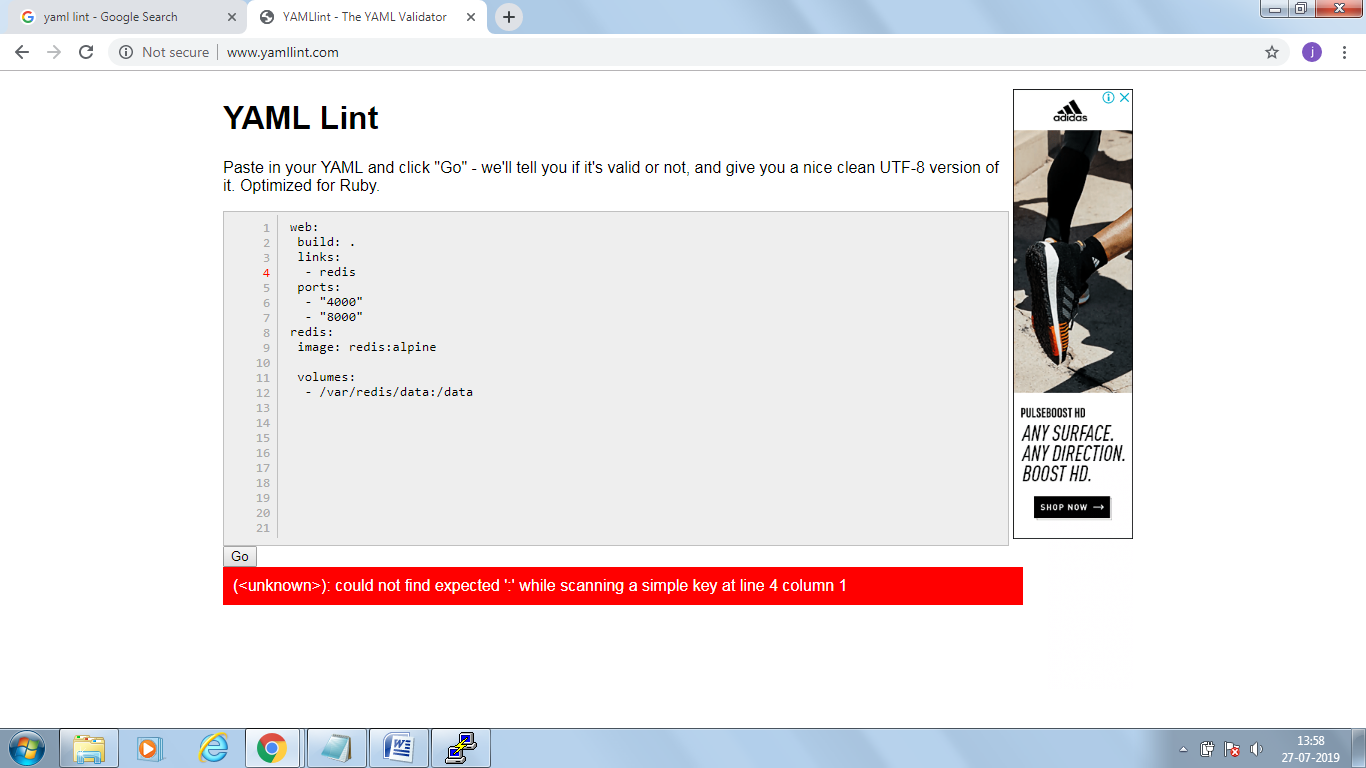
sudo curl -L https://github.com/docker/compose/releases/download/1.24.1/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

docker-compose –version





Vi Dockerfile

---

FROM nginx

MAINTAINER vishal

EXPOSE 80

------------------------

Vi docker-compose.yaml

---

redis:

image: "redis:alpine"

volumes:

- "/var/redis/data:/data"

web:

build: "."

links:

- redis

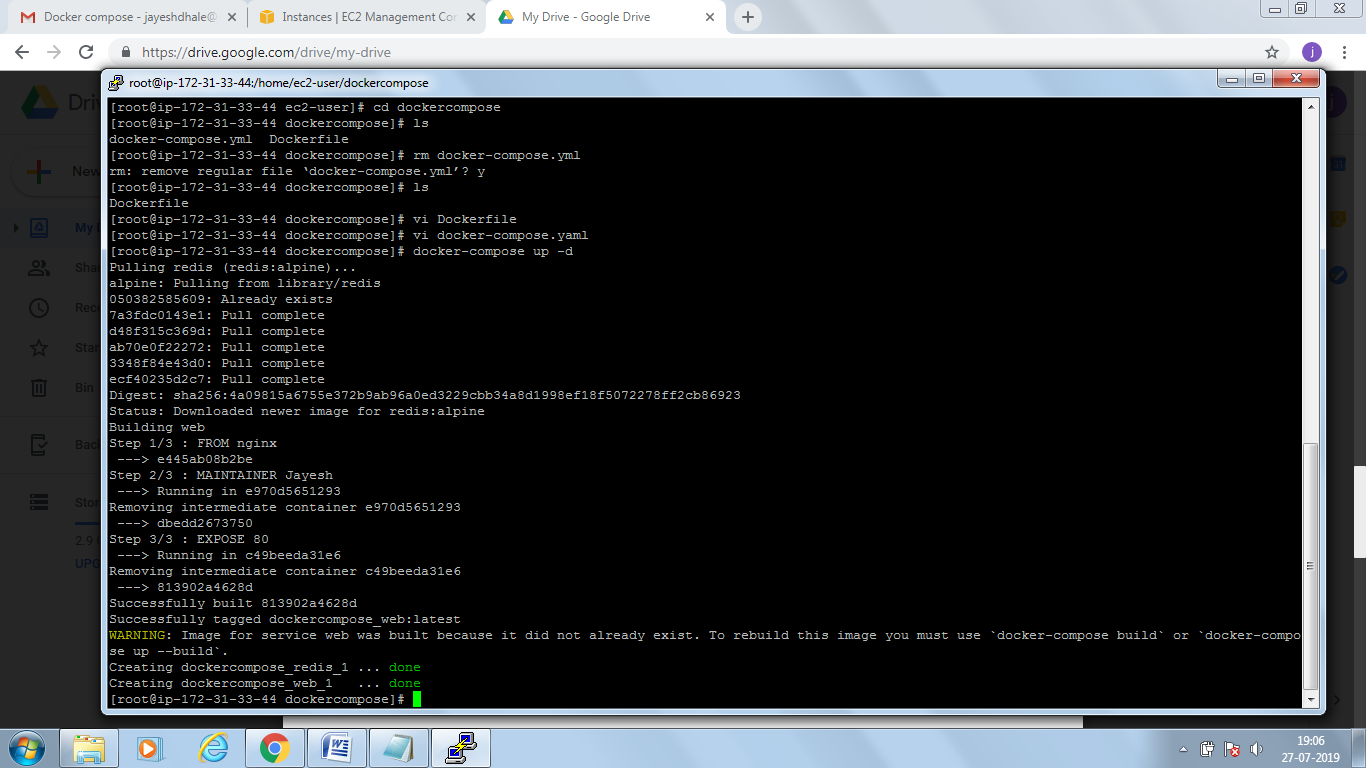
ports:

- "80"

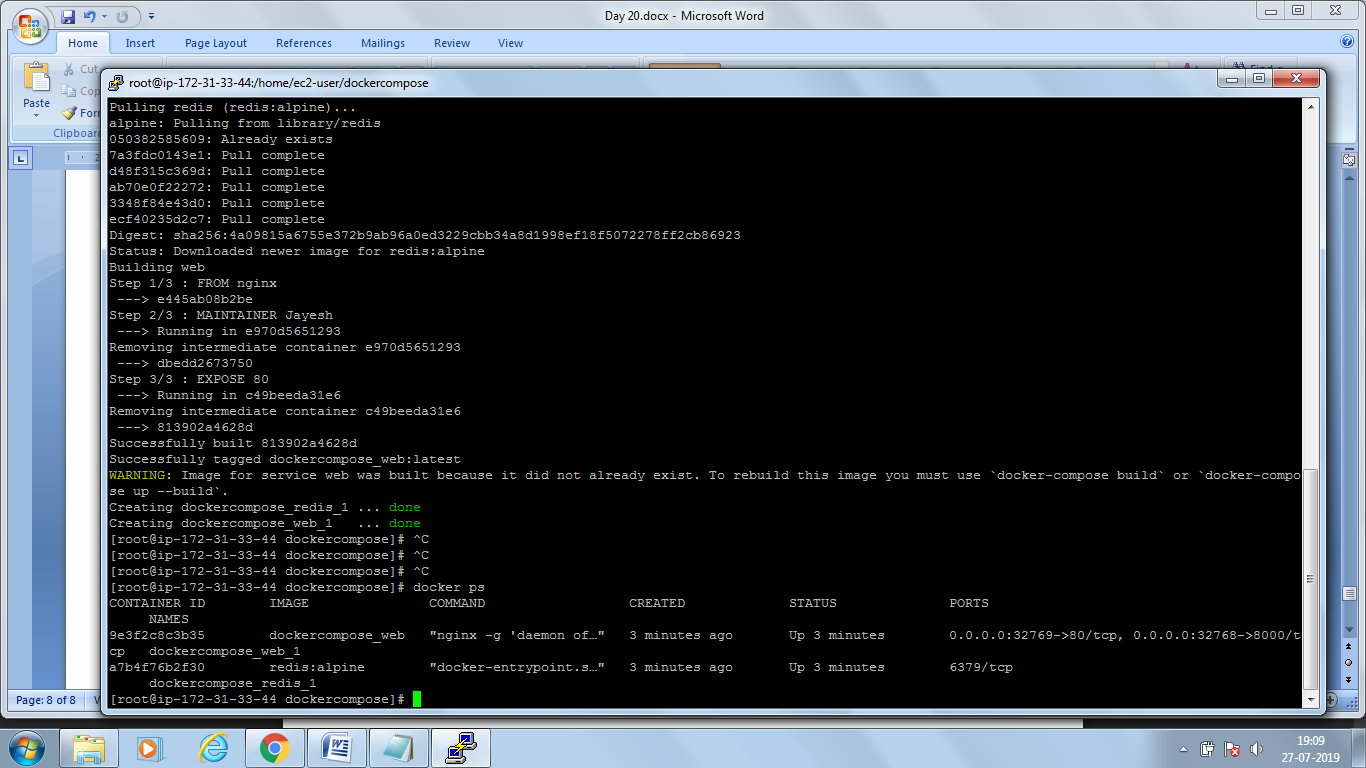
- "8000"

-------------

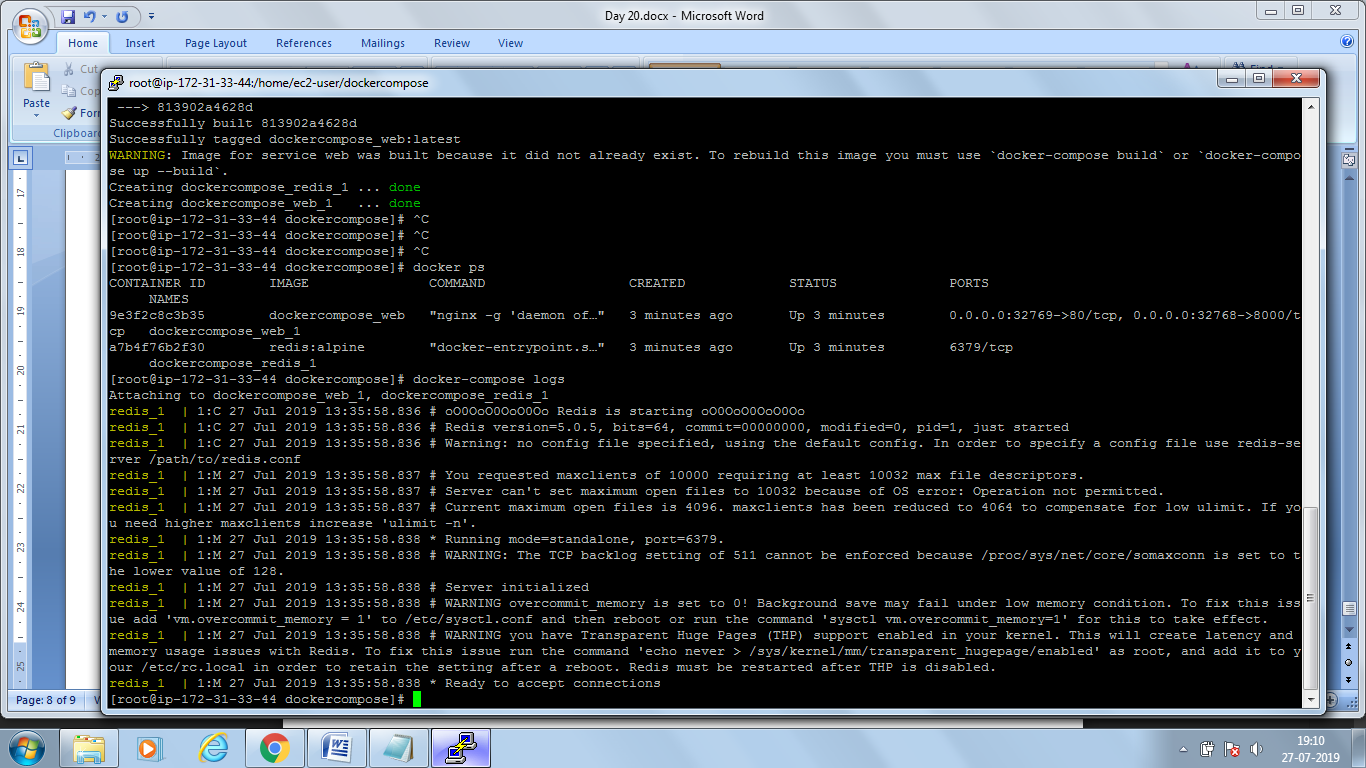
docker-compose up -d



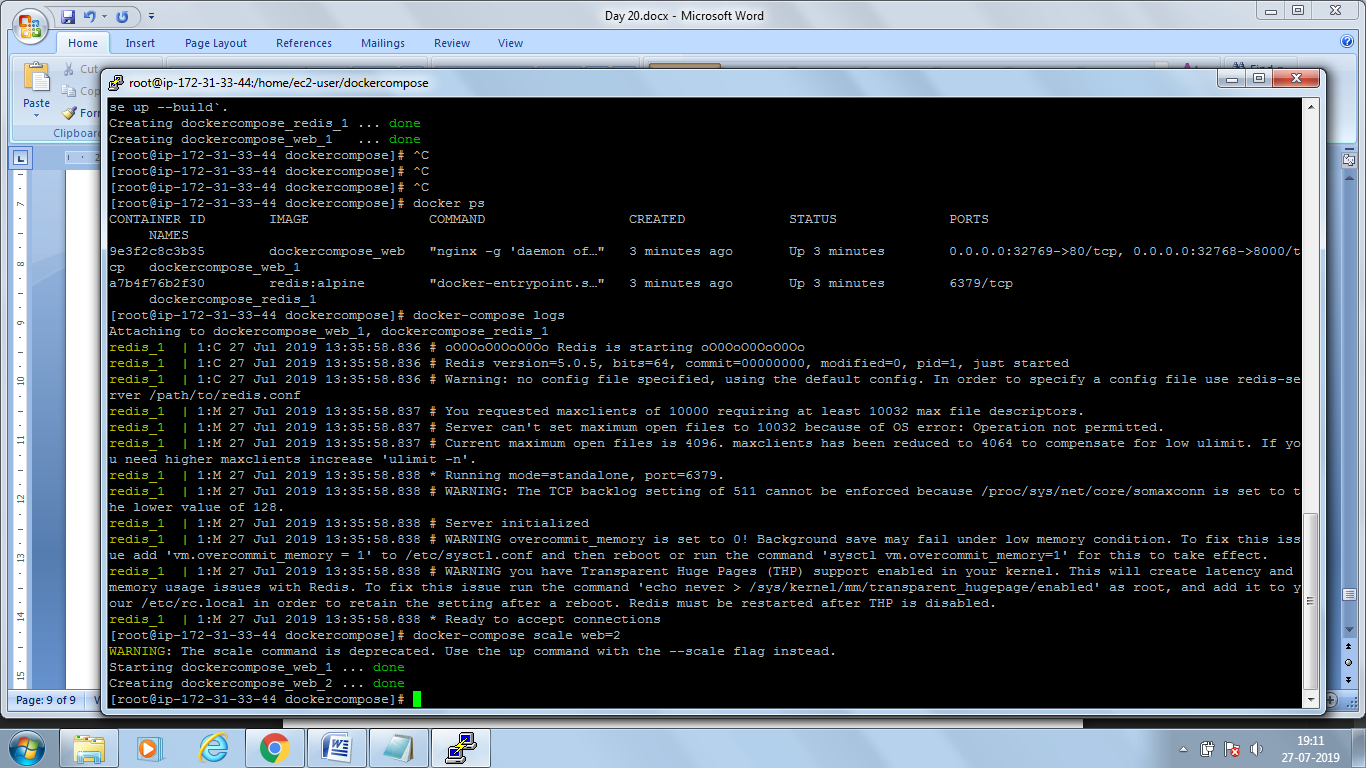
Docker ps



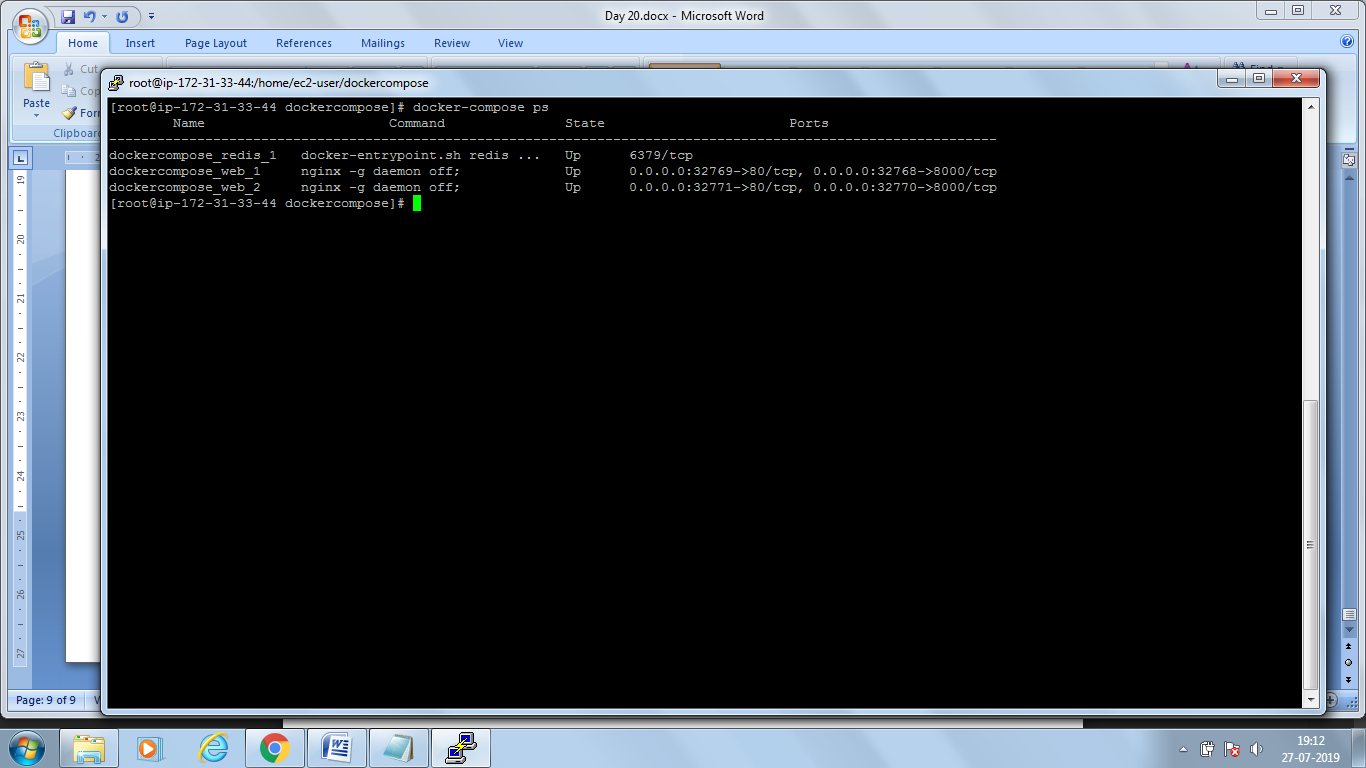
docker-compose logs



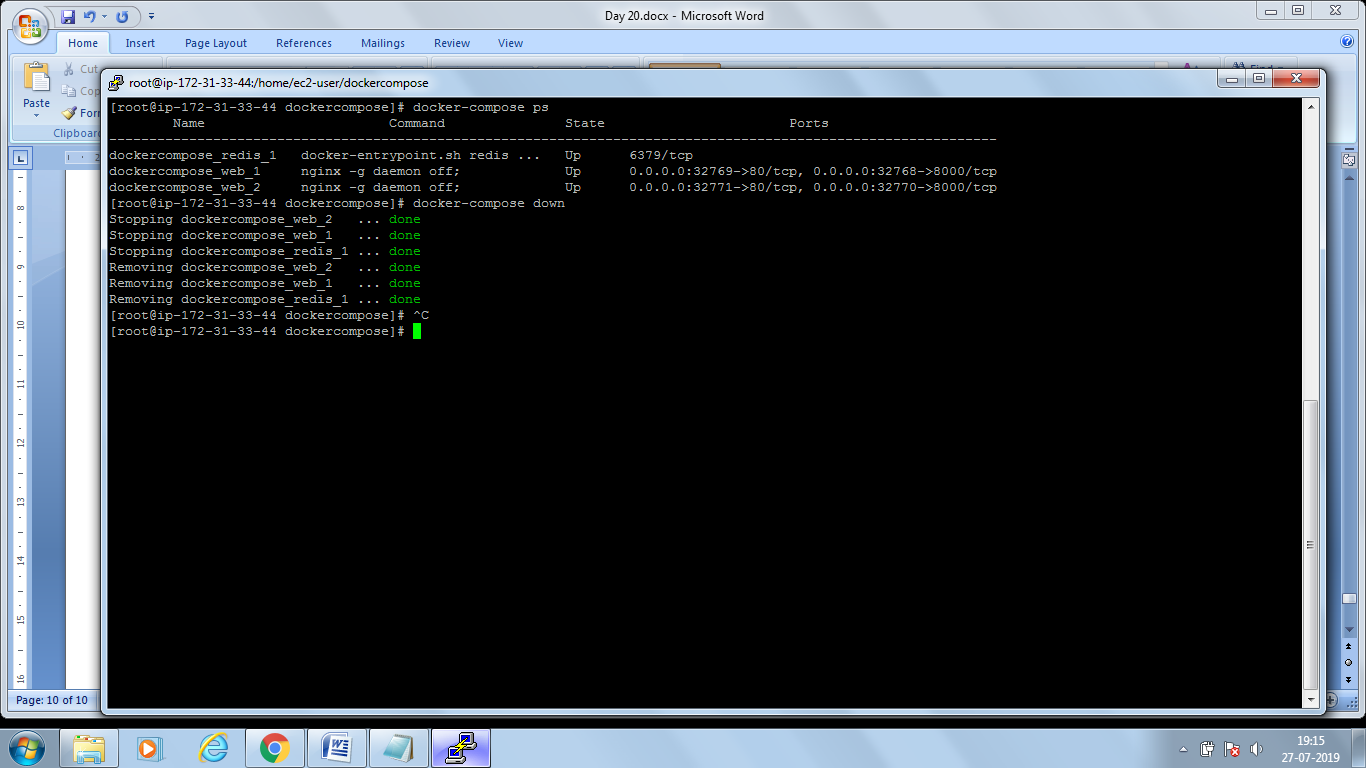
docker-compose scale web=2



docker-compose ps



docker-compose down



[docker compose vs Dockerfile](http://v0cdocker.blogspot.com/2017/11/docker-compose-vs-dockerfile.html)

**Docker** creates isolated machine (container). Each container contains only one process (Apache or Mysql or another); And Dockerfile defines how to build a image.

**Docker compose** allows run, links and configure the **bunch** of containers together.

In your case apache needs to know "where" a mysql. And mysql needs to be waked up before you run apache container.

Dockerfile defines how to create app image. App image contains you application and web-browser.

FROM apache:php5.6

ADD /src /var/www/awesome\_project #add a project src code

ADD /config/apache/awesome\_project.conf /etc/apache2/sites-available/ # add a configuration

# make any things

Then you need to build image docker build my\_app:latest .

At this point you have created image, and you need to run app and links it to db

you have 2 ways to do this:

1) *Native docker approach*. you run db container

docker run --name some-mysql -e MYSQL\_ROOT\_PASSWORD=my-secret-pw -d mysql:latest

and after you need to run app container (image was created before)

docker run --name my\_app --link some-mysql:mysql -P -d my\_app

at this point we have worked application. Bit this simple thing cause us make 2 long command. If you need copy application to another machine you need to repeat this command exactly.

2) *docker-compose way* allows create a configuration for running the containers. It described how exactly run containers.

Simple docker-compose.yml config illustrate this approach

db:

image: mysql

environment:

- MYSQL\_USER=root

- MYSQL\_PASSWORD=root

app:

image: my\_app:latest

ports:

- 80:80

depends\_on:

- db

environment:

# Database

- DB\_USER\_NAME=root

- DB\_USER\_PASSWORD=root

This config allows you run 2 container together, links and configure them.

This is very easy example. and pros of using docker compose not apparent, but if you have 5+ containers it is too hard to run them together without compose.