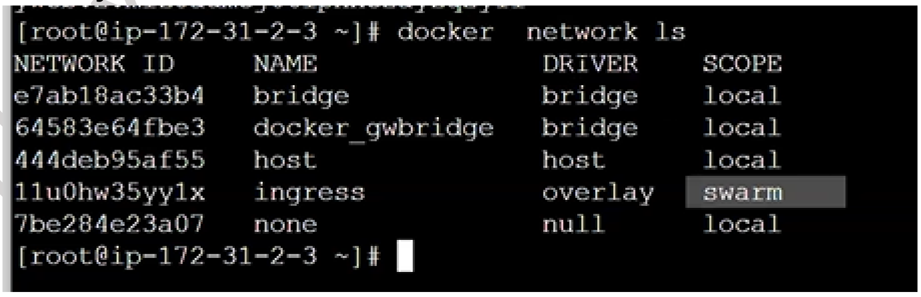
Summary

Session No – 17

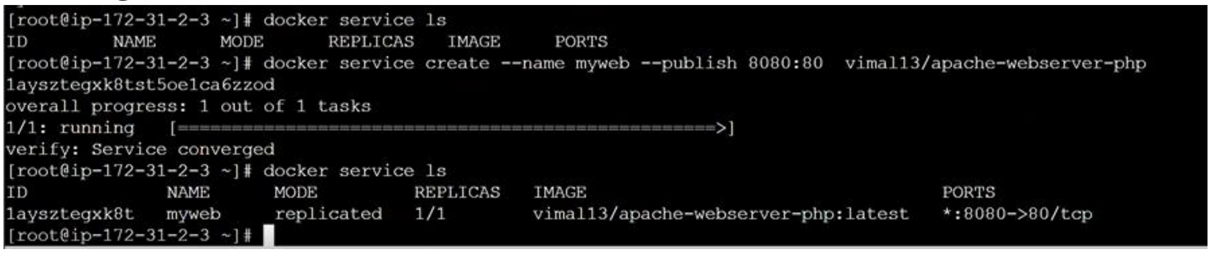
- We can launch the container with
  - Docker run command
  - Docker compose command
  - Swarm cluster
- The docker engine will give the runtime to the container with the help of a program called run c
- By default, docker gives networking in such a way that all the containers running on the same docker host can connect to each other. This kind of networking is called the bridge network
- One issue we face in networking is if we try to connect containers running on a different host, we don't have connectivity
- In multi-tier architecture, we are running our different services on the containers which are running on different host
- The overlay network & overlay driver has the capability to work as a single network on the distributed host
- As soon as we create the cluster in Swarm it automatically creates the overlay network for us & as we join more nodes automatically our distributed network will expand.
- The master node also works as a worker node
- If we list the network on the node, we can see swarm has created the overlay network



```
[root@ip-172-31-2-3 ~]# docker  network ls
NETWORK ID      NAME               DRIVER    SCOPE
e7ab18ac33b4    bridge             bridge    local
64583e64fbe3    docker_gwbridge    bridge    local
444deb95af55    host               host      local
11u0hw35yy1x    ingress            overlay   swarm
7be284e23a07    none               null      local
[root@ip-172-31-2-3 ~]#
```

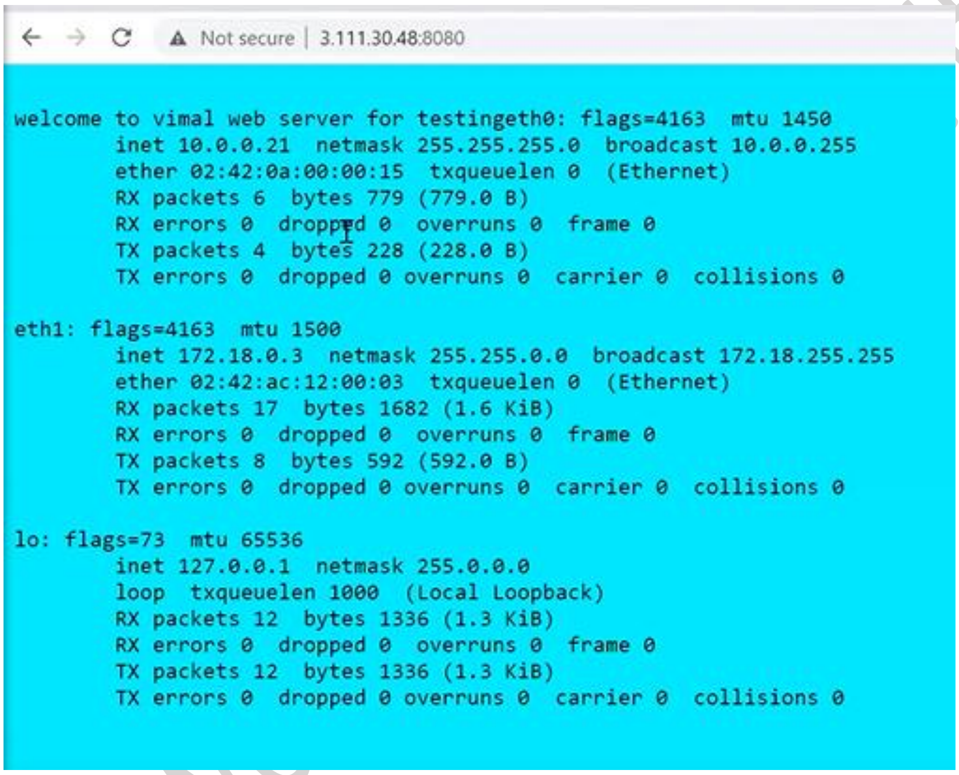
LinuxWorld Informatics pvt ltd

- Creating service on the master node



```
[root@ip-172-31-2-3 ~]# docker service ls
ID        NAME      MODE      REPLICAS   IMAGE     PORTS
[root@ip-172-31-2-3 ~]# docker service create --name myweb --publish 8080:80  vimal13/apache-webserver-php
1aysztegxk8tst5oe1ca6zzod
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
[root@ip-172-31-2-3 ~]# docker service ls
ID             NAME      MODE         REPLICAS   IMAGE                                 PORTS
1aysztegxk8t   myweb     replicated   1/1        vimal13/apache-webserver-php:latest   *:8080->80/tcp
[root@ip-172-31-2-3 ~]#
```

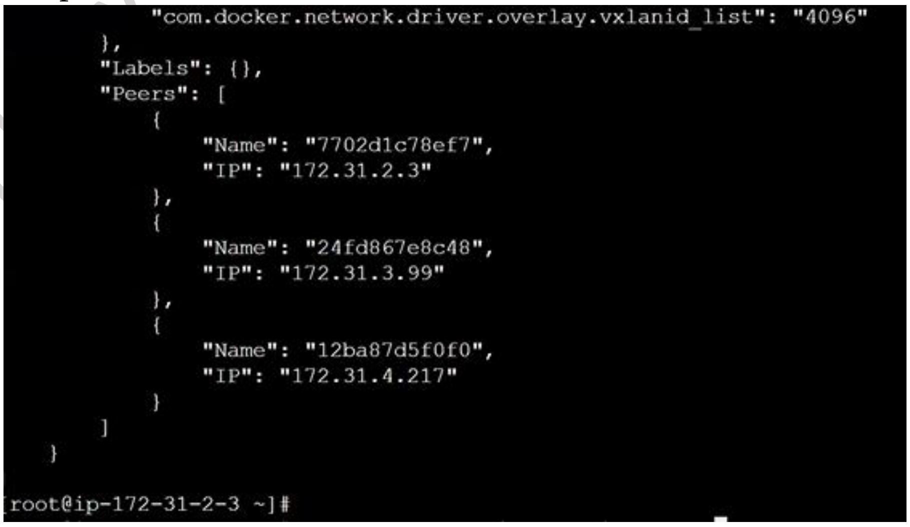
- The interesting thing is our container is running on the worker node & if we try to connect the website with the master's IP address, we can see the website



Not secure | 3.111.30.48:8080

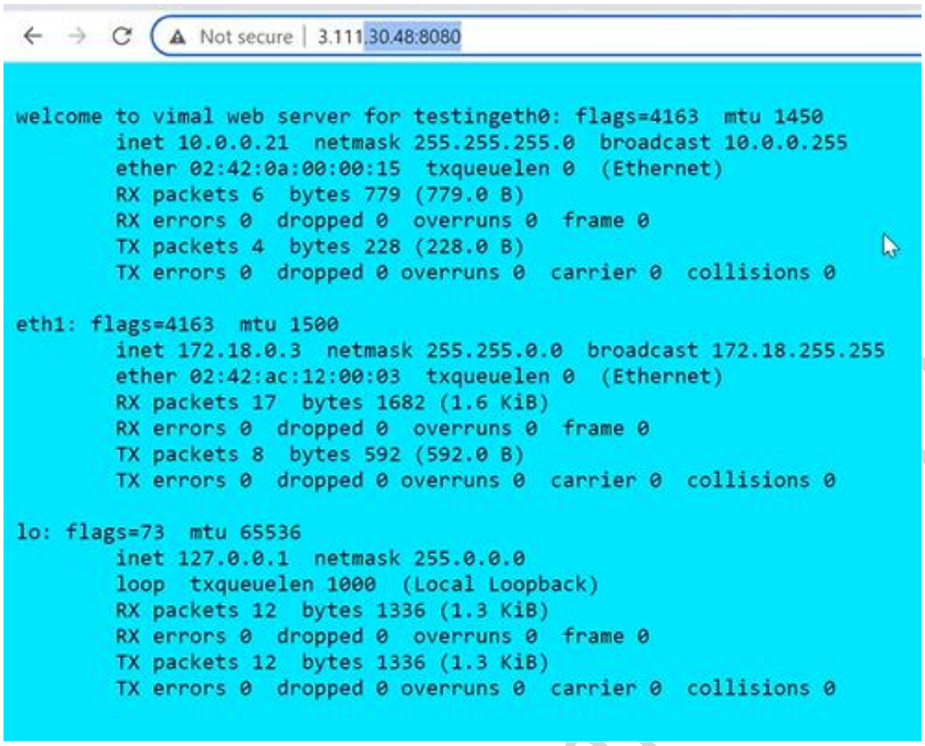
```
welcome to vimal web server for testingeth0: flags=4163  mtu 1450
        inet 10.0.0.21  netmask 255.255.255.0  broadcast 10.0.0.255
        ether 02:42:0a:00:00:15  txqueuelen 0  (Ethernet)
        RX packets 6  bytes 779 (779.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4  bytes 228 (228.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163  mtu 1500
        inet 172.18.0.3  netmask 255.255.0.0  broadcast 172.18.255.255
        ether 02:42:ac:12:00:03  txqueuelen 0  (Ethernet)
        RX packets 17  bytes 1682 (1.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 592 (592.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 12  bytes 1336 (1.3 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 12  bytes 1336 (1.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

- If we inspect the overlay network, we can see the network is maintaining the peers or nodes



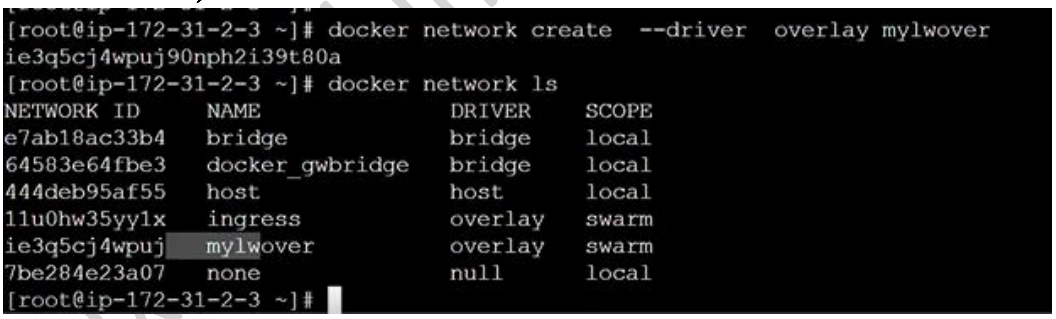
```
            "com.docker.network.driver.overlay.vxlanid_list": "4096"
        },
        "Labels": {},
        "Peers": [
            {
                "Name": "7702d1c78ef7",
                "IP": "172.31.2.3"
            },
            {
                "Name": "24fd867e8c48",
                "IP": "172.31.3.99"
            },
            {
                "Name": "12ba87d5f0f0",
                "IP": "172.31.4.217"
            }
        ]
    }

[root@ip-172-31-2-3 ~]#
```

- Even if we try to connect to the website with the worker node’s IP address, we can see the website



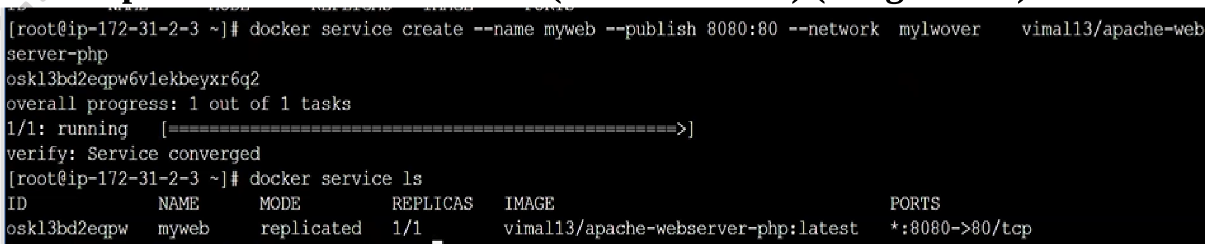
```
Not secure | 3.111.30.48:8080

welcome to vimal web server for testingeth0: flags=4163  mtu 1450
        inet 10.0.0.21  netmask 255.255.255.0  broadcast 10.0.0.255
        ether 02:42:0a:00:00:15  txqueuelen 0  (Ethernet)
        RX packets 6  bytes 779 (779.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4  bytes 228 (228.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163  mtu 1500
        inet 172.18.0.3  netmask 255.255.0.0  broadcast 172.18.255.255
        ether 02:42:ac:12:00:03  txqueuelen 0  (Ethernet)
        RX packets 17  bytes 1682 (1.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 592 (592.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 12  bytes 1336 (1.3 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 12  bytes 1336 (1.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

- Creating own personal overlay network with the docker network command
  - Command: - **docker network create --driver overlay (network name)**



```
[root@ip-172-31-2-3 ~]# docker network create  --driver  overlay mylwover
ie3q5cj4wpuj90nph2i39t80a
[root@ip-172-31-2-3 ~]# docker network ls
NETWORK ID      NAME              DRIVER    SCOPE
e7ab18ac33b4    bridge            bridge    local
64583e64fbe3    docker_gwbridge   bridge    local
444deb95af55    host              host      local
11u0hw35yy1x    ingress           overlay   swarm
ie3q5cj4wpuj    mylwover          overlay   swarm
7be284e23a07    none              null      local
[root@ip-172-31-2-3 ~]#
```

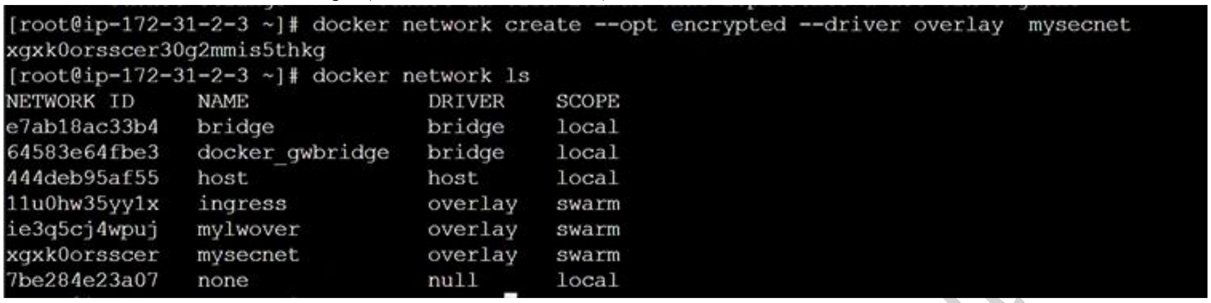
- Launching service in a custom overlay network
  - Command: - **docker service create --name (service name) --publish 8080:80 --network (network name) (image name)**



```
[root@ip-172-31-2-3 ~]# docker service create --name myweb --publish 8080:80 --network  mylwover    vimal13/apache-web
server-php
oskl3bd2eqpw6v1ekbeyxr6q2
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
[root@ip-172-31-2-3 ~]# docker service ls
ID             NAME      MODE         REPLICAS   IMAGE                                  PORTS
oskl3bd2eqpw   myweb     replicated   1/1        vimal13/apache-webserver-php:latest    *:8080->80/tcp
```

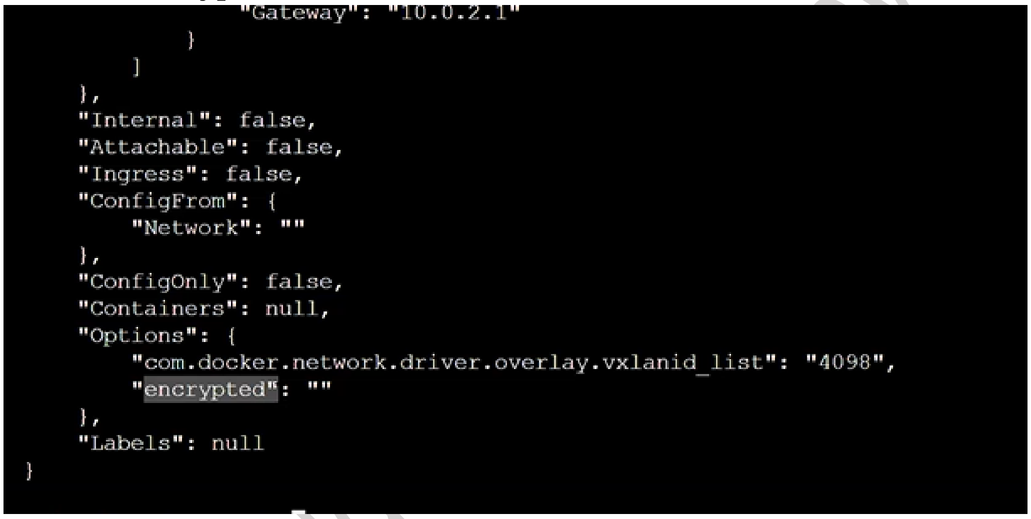
- Whenever we create an overlay network or distributed network the network packet between the container is going in a clear text

- With the **encrypted** keyword in the docker network command, we can encrypt the connection
  - Command :- **docker network create --opt encrypted --driver overlay (network name)**



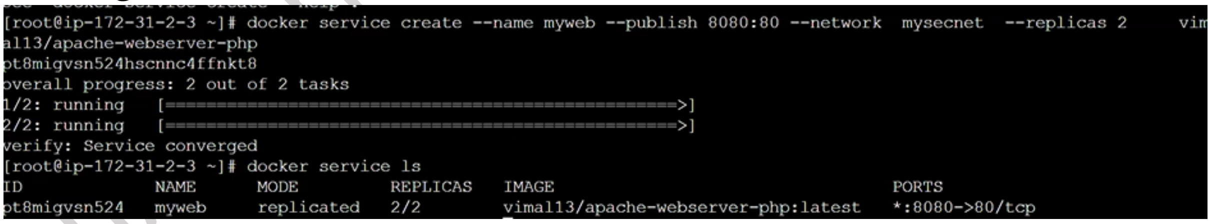
```
[root@ip-172-31-2-3 ~]# docker network create --opt encrypted --driver overlay  mysecnet
xgxk0orsscer30g2mmis5thkg
[root@ip-172-31-2-3 ~]# docker network ls
NETWORK ID     NAME              DRIVER    SCOPE
e7ab18ac33b4   bridge            bridge    local
64583e64fbe3   docker_gwbridge   bridge    local
444deb95af55   host              host      local
11u0hw35yy1x   ingress           overlay   swarm
ie3q5cj4wpuj   mylwover          overlay   swarm
xgxk0orsscer   mysecnet          overlay   swarm
7be284e23a07   none              null      local
```

- If we inspect the network we can see the communication between the two nodes is encrypted



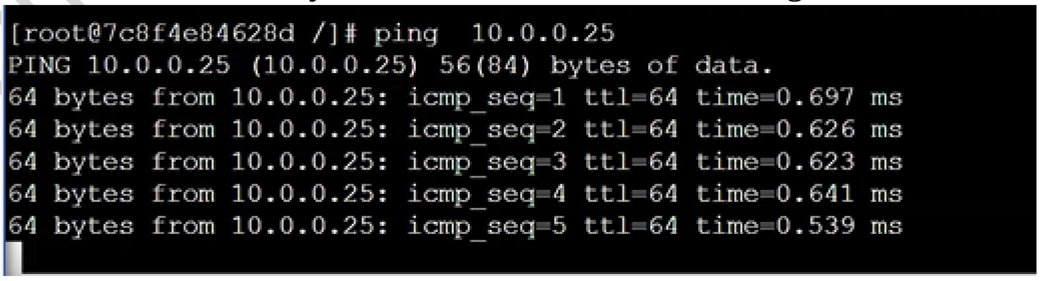
```
                    "Gateway": "10.0.2.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": null,
        "Options": {
            "com.docker.network.driver.overlay.vxlanid_list": "4098",
            "encrypted": ""
        },
        "Labels": null
    }
```

- Creating service in a secure network



```
[root@ip-172-31-2-3 ~]# docker service create --name myweb --publish 8080:80 --network  mysecnet  --replicas 2     vim
al13/apache-webserver-php
pt8migvsn524hscnnc4ffnkt8
overall progress: 2 out of 2 tasks
1/2: running   [==================================================>]
2/2: running   [==================================================>]
verify: Service converged
[root@ip-172-31-2-3 ~]# docker service ls
ID             NAME      MODE         REPLICAS   IMAGE                                PORTS
pt8migvsn524   myweb     replicated   2/2        vimal13/apache-webserver-php:latest   *:8080->80/tcp
```

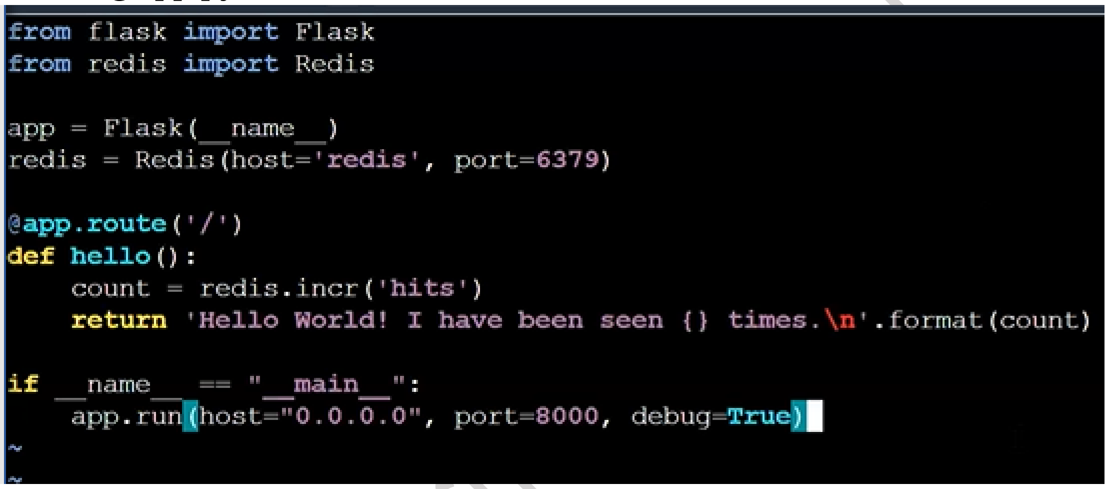
- We have connectivity between the containers running on a different node



```
[root@7c8f4e84628d /]# ping  10.0.0.25
PING 10.0.0.25 (10.0.0.25) 56(84) bytes of data.
64 bytes from 10.0.0.25: icmp_seq=1 ttl=64 time=0.697 ms
64 bytes from 10.0.0.25: icmp_seq=2 ttl=64 time=0.626 ms
64 bytes from 10.0.0.25: icmp_seq=3 ttl=64 time=0.623 ms
64 bytes from 10.0.0.25: icmp_seq=4 ttl=64 time=0.641 ms
64 bytes from 10.0.0.25: icmp_seq=5 ttl=64 time=0.539 ms
```

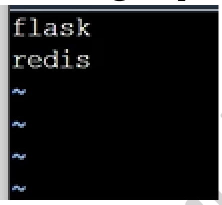
- For encryption in the overlay network, it is using IPSEC protocol

- Every network card has its own physical address known as a MAC address because of which we have LAN connectivity. But we are extending the capability of Lan this concept is called VXLAN
- Docker compose has a limitation it works only on a single node
- Stack is one capability of docker with its help we can launch the container in different nodes and automate the docker-compose file
- Docker stack practical
  - Creating app.py file



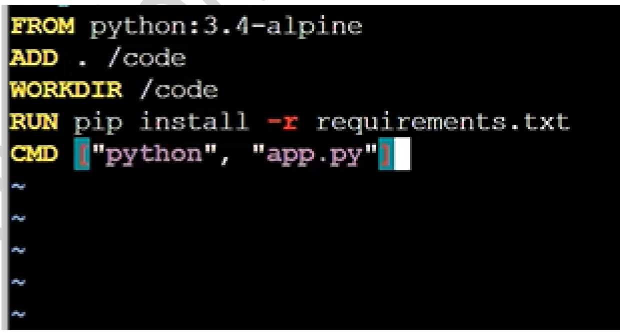
```
from flask import Flask
from redis import Redis

app = Flask(__name__)
redis = Redis(host='redis', port=6379)

@app.route('/')
def hello():
    count = redis.incr('hits')
    return 'Hello World! I have been seen {} times.\n'.format(count)

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8000, debug=True)
~
~
```

  - Creating requirement.txt



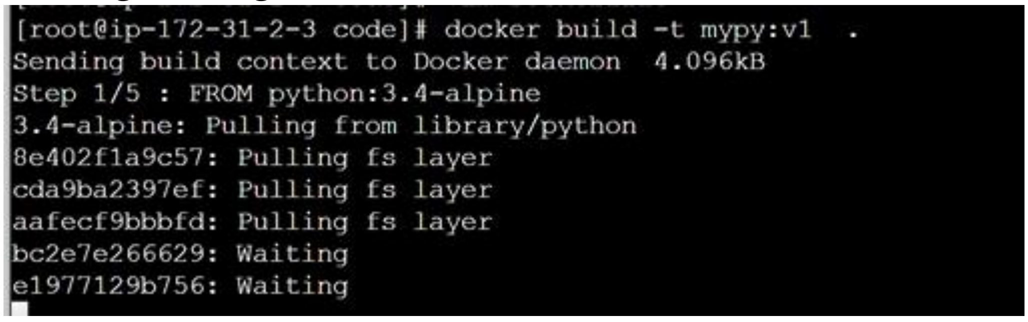
```
flask
redis
~
~
~
~
```

  - Docker file



```
FROM python:3.4-alpine
ADD . /code
WORKDIR /code
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
~
~
~
~
~
```

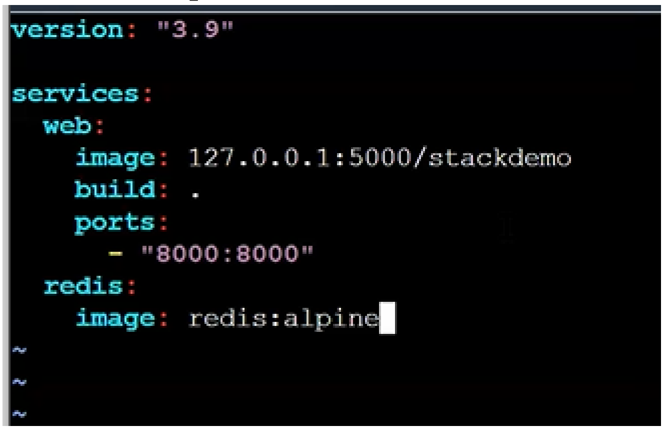
  - Creating the image



```
[root@ip-172-31-2-3 code]# docker build -t mypy:v1  .
Sending build context to Docker daemon  4.096kB
Step 1/5 : FROM python:3.4-alpine
3.4-alpine: Pulling from library/python
8e402f1a9c57: Pulling fs layer
cda9ba2397ef: Pulling fs layer
aafecf9bbbfd: Pulling fs layer
bc2e7e266629: Waiting
e1977129b756: Waiting
```

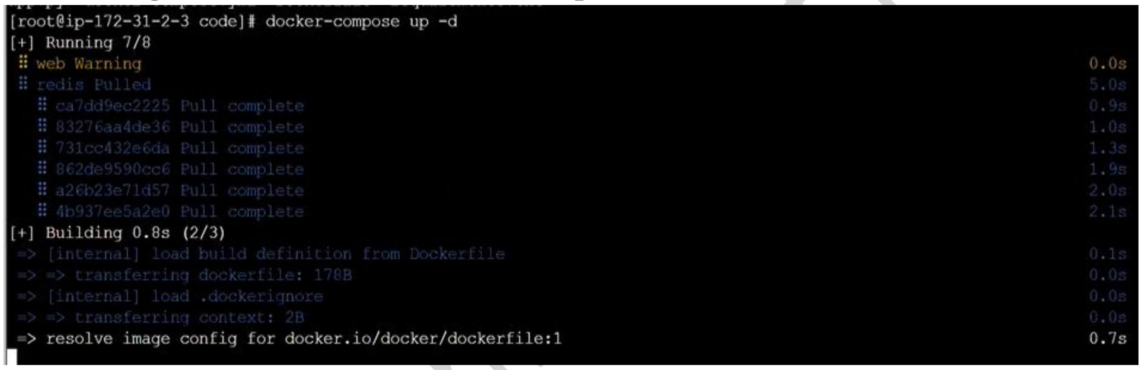
Linux World Inform

- Docker-compose file



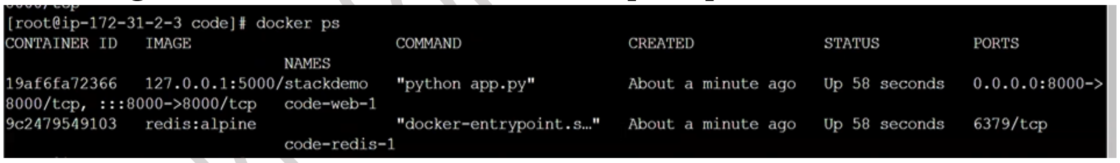
```
version: "3.9"

services:
  web:
    image: 127.0.0.1:5000/stackdemo
    build: .
    ports:
      - "8000:8000"
  redis:
    image: redis:alpine
~
~
~
```

- Launching container with docker-compose



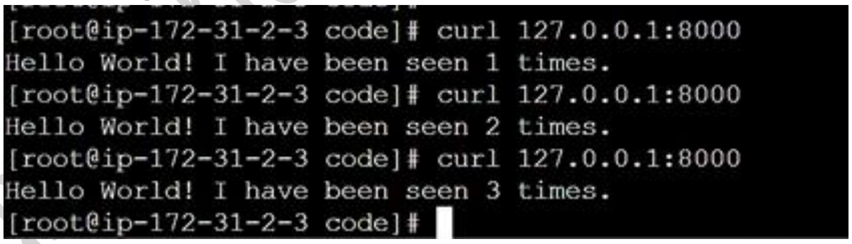
```
[root@ip-172-31-2-3 code]# docker-compose up -d
[+] Running 7/8
 ⠿ web Warning                                                                    0.0s
 ⠿ redis Pulled                                                                   5.0s
   ⠿ ca7dd9ec2225 Pull complete                                                   0.9s
   ⠿ 83276aa4de36 Pull complete                                                   1.0s
   ⠿ 731cc432e6da Pull complete                                                   1.3s
   ⠿ 862de9590cc6 Pull complete                                                   1.9s
   ⠿ a26b23e71d57 Pull complete                                                   2.0s
   ⠿ 4b937ee5a2e0 Pull complete                                                   2.1s
[+] Building 0.8s (2/3)
 => [internal] load build definition from Dockerfile                              0.1s
 => => transferring dockerfile: 178B                                              0.0s
 => [internal] load .dockerignore                                                 0.0s
 => => transferring context: 2B                                                   0.0s
 => resolve image config for docker.io/docker/dockerfile:1                        0.7s
```

- Checking containers with the docker-compose ps command



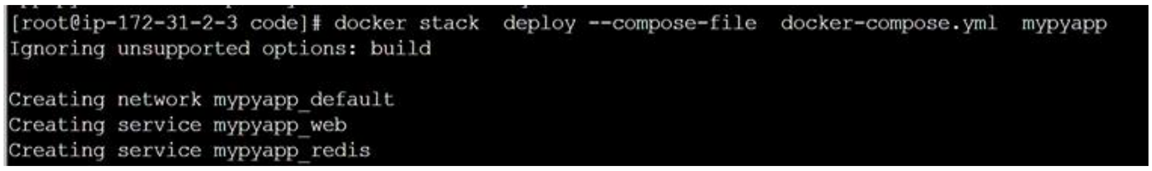
```
[root@ip-172-31-2-3 code]# docker ps
CONTAINER ID   IMAGE                      COMMAND                  CREATED              STATUS          PORTS
                                     NAMES
19af6fa72366   127.0.0.1:5000/stackdemo   "python app.py"          About a minute ago   Up 58 seconds   0.0.0.0:8000->
8000/tcp, :::8000->8000/tcp   code-web-1
9c2479549103   redis:alpine               "docker-entrypoint.s…"   About a minute ago   Up 58 seconds   6379/tcp
                                     code-redis-1
```

- Accessing the website with the curl command



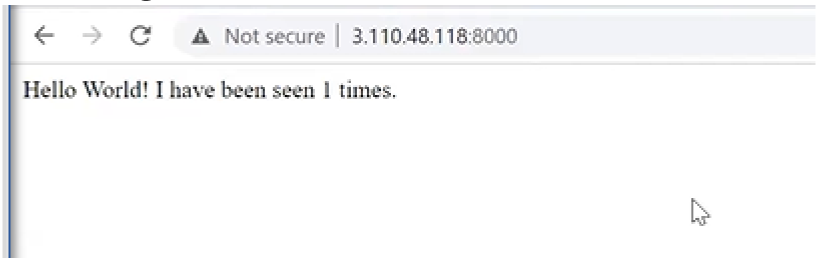
```
[root@ip-172-31-2-3 code]# curl 127.0.0.1:8000
Hello World! I have been seen 1 times.
[root@ip-172-31-2-3 code]# curl 127.0.0.1:8000
Hello World! I have been seen 2 times.
[root@ip-172-31-2-3 code]# curl 127.0.0.1:8000
Hello World! I have been seen 3 times.
[root@ip-172-31-2-3 code]#
```

- Deploying stack
  - Command:- **docker stack deploy --compose-file docker-compose.yml (Name for stack)**



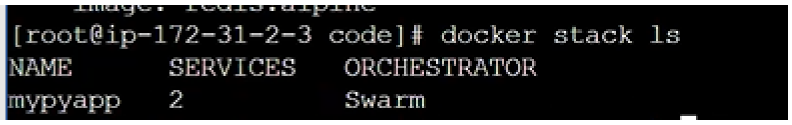
```
[root@ip-172-31-2-3 code]# docker stack deploy --compose-file docker-compose.yml mypyapp
Ignoring unsupported options: build

Creating network mypyapp_default
Creating service mypyapp_web
Creating service mypyapp_redis
```

- Accessing the website from the browser



```
Not secure | 3.110.48.118:8000
Hello World! I have been seen 1 times.
```

- **docker stack ls** command is used to list the stack



```
[root@ip-172-31-2-3 code]# docker stack ls
NAME       SERVICES   ORCHESTRATOR
mypyapp    2          Swarm
```

- Docker stack has the capability to launch in Kubernetes cluster also

Important link:

https://docs.docker.com/engine/swarm/stack-deploy/

Linux World Informatics pvt ltd