IZMIR

@dockerizmir

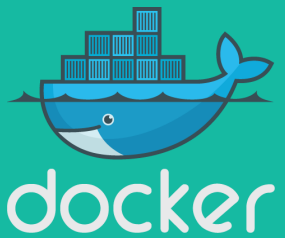
Build, Ship, Run

Docker is the world's leading software containerization platform

Demystifying Docker Swarm Mode

Ajeet Singh Raina

Docker Captain – Docker, Inc.



Who Am I?

- Sr. Systems Development Engineer at DellEMC
- 1st half of my career was in CGI & VMware
- 2nd half of my career has been in System Integration
- Testing/Project Lead for Dell EMC.
- Definitely more IT pro than developer
- @ajeetsraina (a frequent Twitterati)

The {code} Catalyst Program!

http://www.collabnix.com

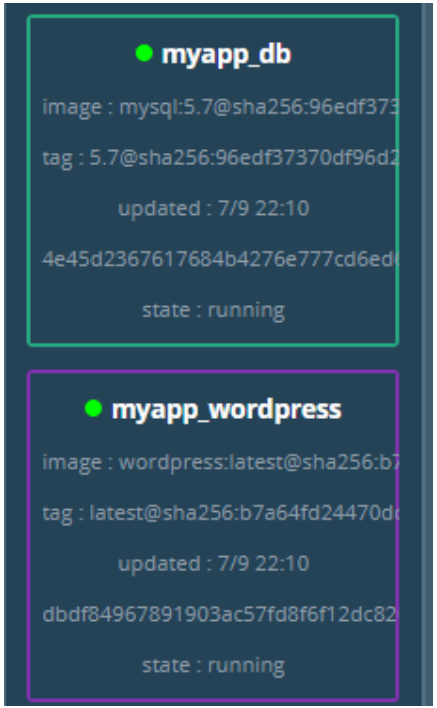
Agenda

- Introduction to Docker Swarm
- Docker Swarm Mode Features
- Docker Stack Deployment
- What's new in Docker 17.06 Swarm Mode - Hybrid Swarm Setup, Toplogy Scheduling
- Demo – Hybrid Swarm(Play with Docker)

Introduction to Docker Swarm Mode

A Little Background: What is Swarm?

Let’s start with Single Docker Host Application

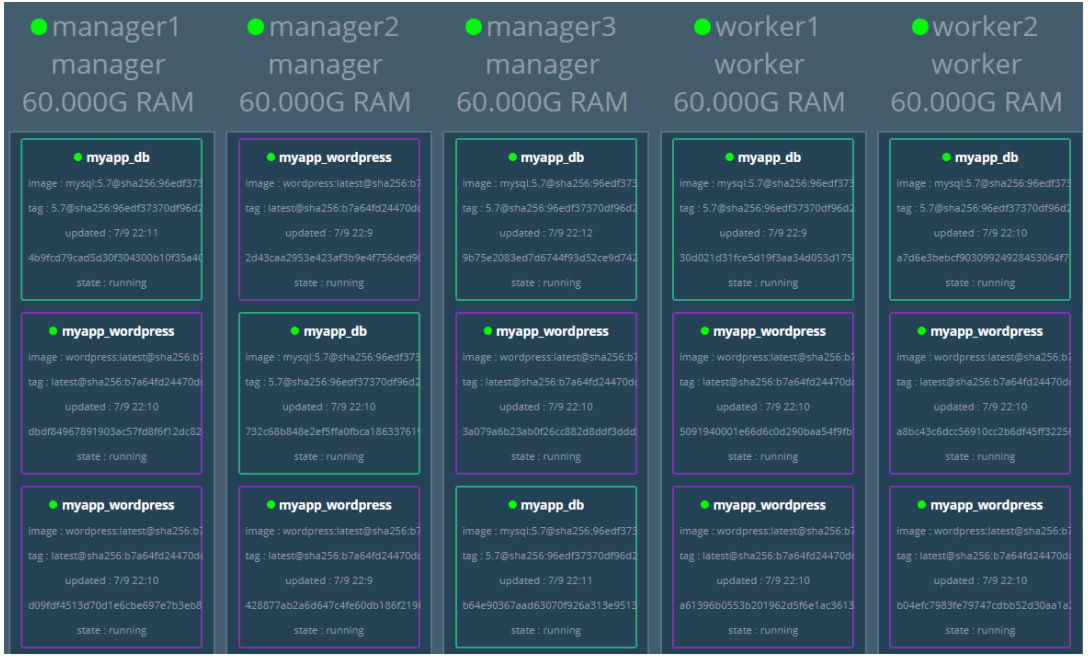


A Docker Host

http://collabnix.com/getting-started-with-docker-swarm/

A Little Background: What is Swarm?

You want to add more hosts..



docker

A Little Background: What is Swarm?

But Wait…

High Availability

Rolling Updates

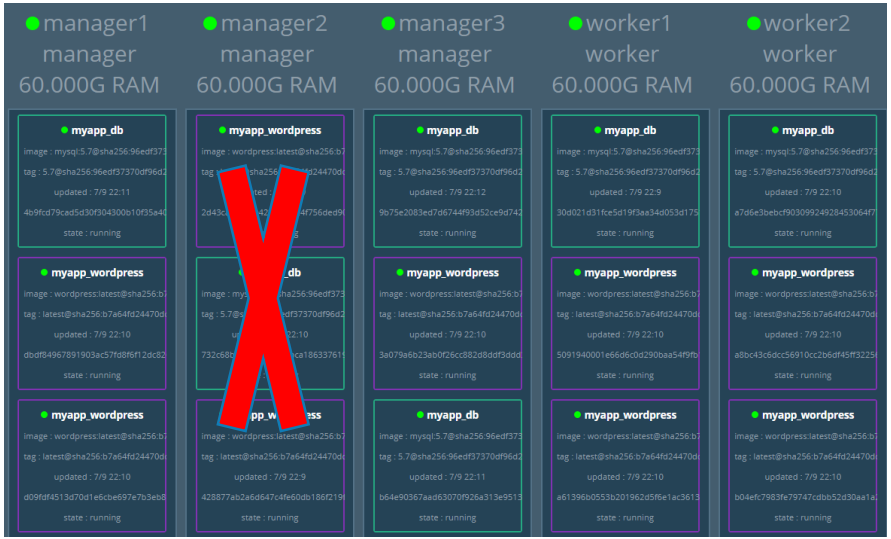
Service Discovery

Scalability

Scheduling

Failure Management

Container Security



docker

Docker Swarm Mode comes to rescue..

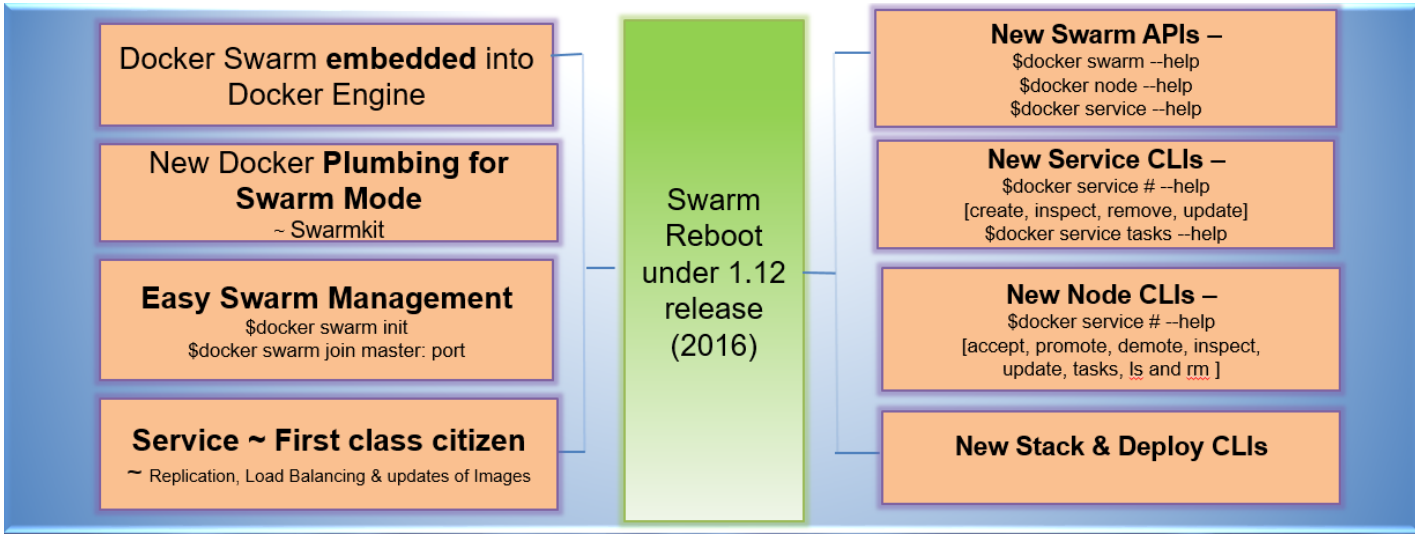
orchestration

http://collabnix.com/new-docker-1-12-comes-with-built-in-distribution-orchestration-system/

docker

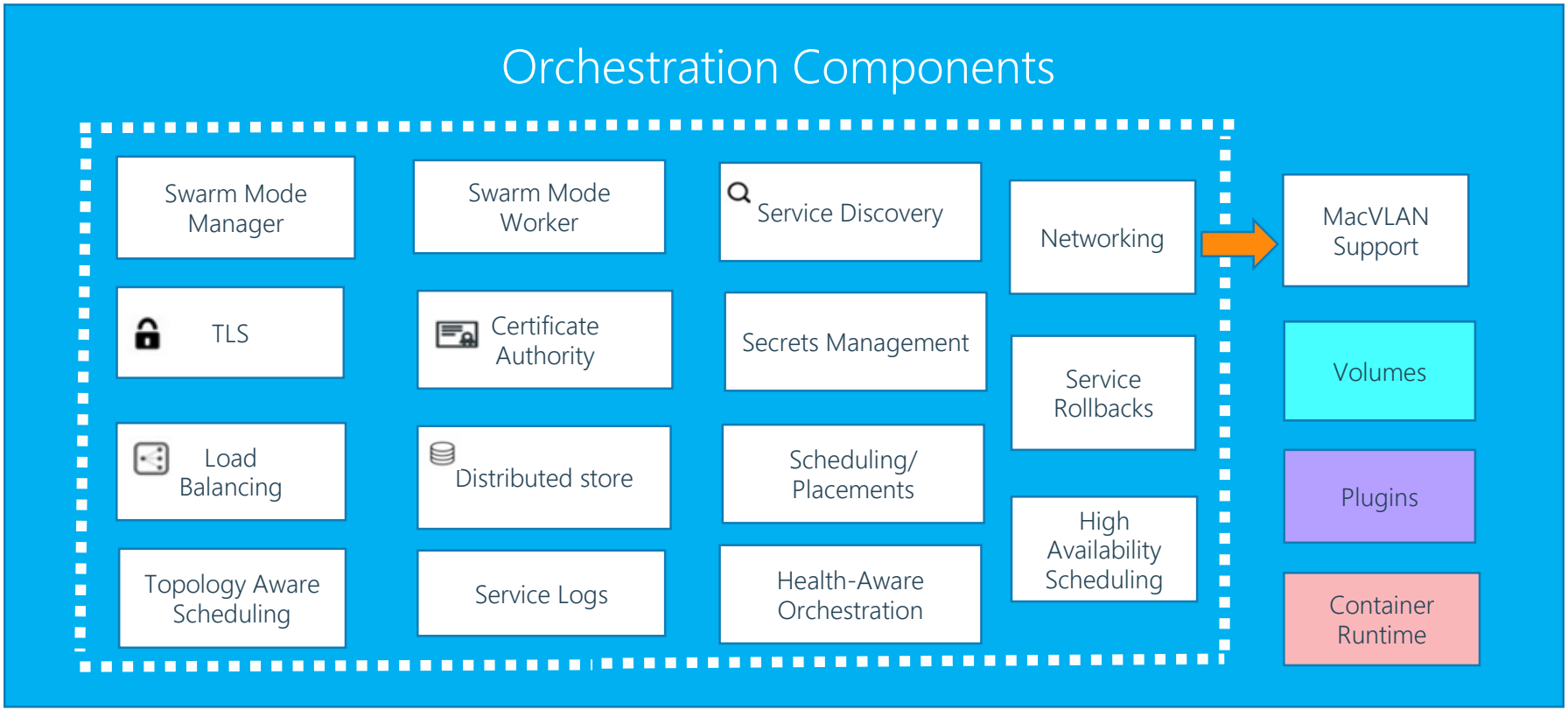
What is Swarm Mode?

- A swarm consists of one or more nodes: physical or virtual machines running Docker Engine.
- It was introduced first under Docker 1.12 release.
- It enables the ability to deploy containers across multiple Docker hosts, using overlay networks for service discovery with a built-in load balancer for scaling the services.



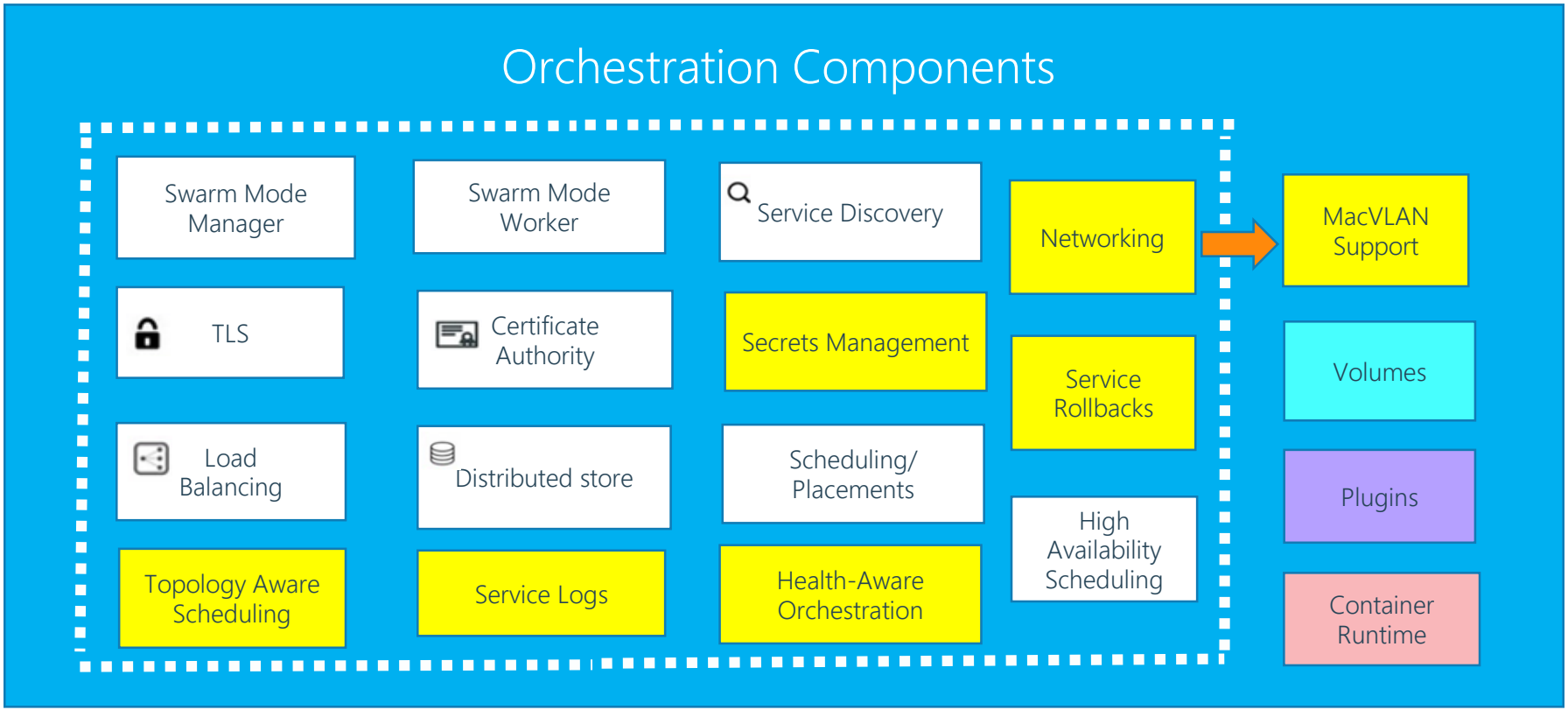
http://collabnix.com/docker-1-12-swarm-mode-under-the-hood/

Swarm Mode Features under Docker 17.06



docker

Swarm Mode Features under Docker 17.06



docker

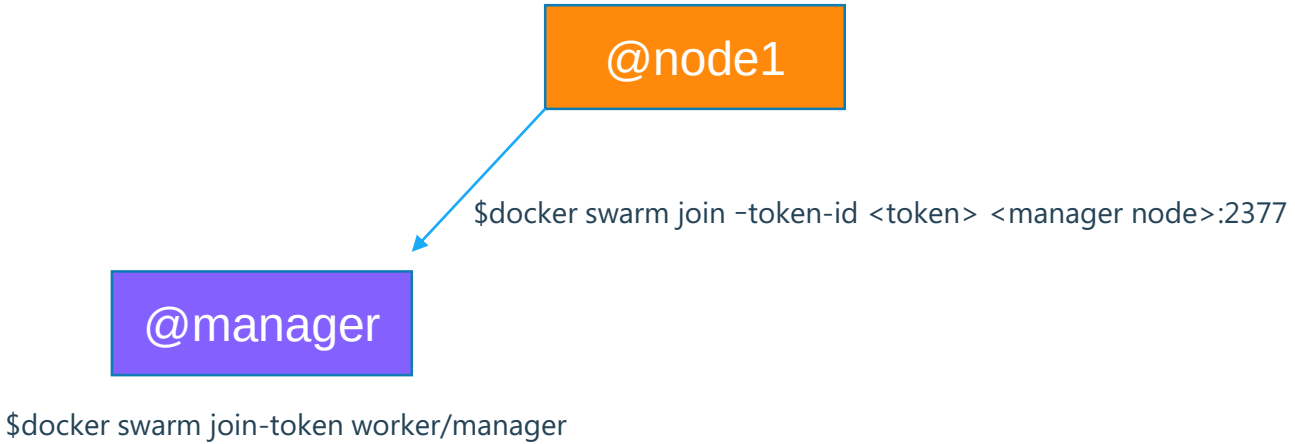
Building a Swarm Topology

Building a Swarm Topology – Manual Way

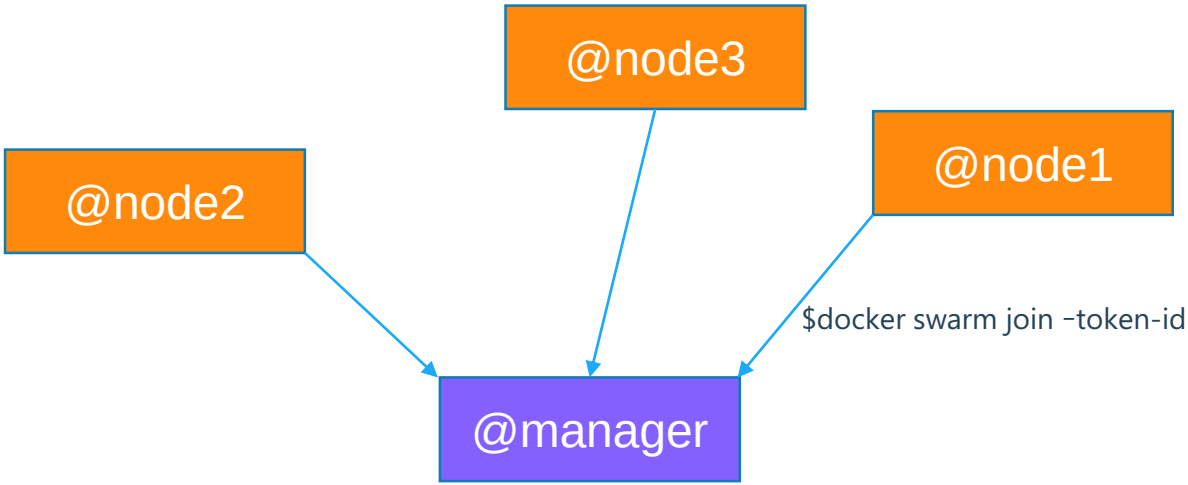
@manager

$docker swarm init --advertise-addr <IP of manager node>:2377
or
$docker swarm init --listen-addr <IP of manager node>:2377

Building a Swarm Topology – Manual Way



Building a Swarm Topology – Manual Way



$docker swarm join –token-id <token> <manager node>:2377

$docker swarm join-token worker/manager

```
$docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS

1wmnidfmcop2uenypd1e8uh5k *   manager             Ready               Active              Leader
9ai9hbfm2ugkmjyt7kpx4hi9k     node1               Ready               Active              Reachable
j8iylu9nxp5zledepqp3shdw7     node2               Ready               Active              Reachable
fd8iylu9nxp5zledepqp3shdw7    node3               Ready               Active              Reachable
```


docker

Building Swarm Topology – Scripted Method(Docker Machine)

```
master_node=master
node01_node=node01
node02_node=node02

# Initialize Virtualbox machines using Docker Machine
for i in $master_node $node01_node $node02_node; do docker-machine create -d virtualbox $i; done

# Create the Docker swarm
docker $(docker-machine config $master_node) swarm init \
--advertise-addr $(docker-machine ip $master_node):2377

# Add workers to the Docker Swarm
docker $(docker-machine config $node01_node) swarm join \
--token `docker $(docker-machine config $master_node) swarm join-token worker -q` \
$(docker-machine ip $master_node):2377

docker $(docker-machine config $node02_node) swarm join \
--token `docker $(docker-machine config $master_node) swarm join-token worker -q` \
$(docker-machine ip $master_node):2377

# Run the Docker Swarm visualizer
docker $(docker-machine config $master_node) run -it -d -p 5000:5000 \
-e HOST=`docker-machine ip $master_node` \
-e PORT=5000 \
-v /var/run/docker.sock:/var/run/docker.sock \
dockersamples/visualizer
```

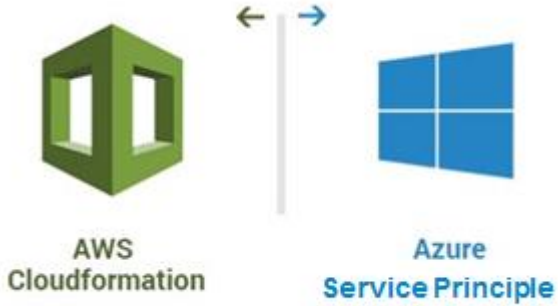
@manager

@node1,2

docker

Building a Swarm Topology – Cloud

Docker for AWS

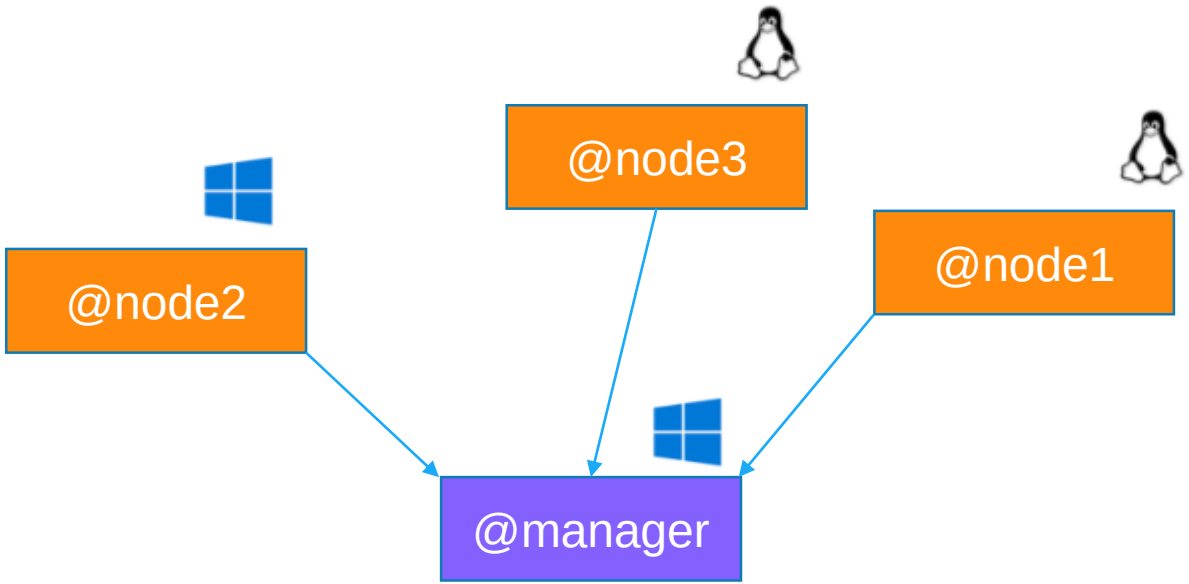


Docker for Azure

Docker for GCP

Deployment manager

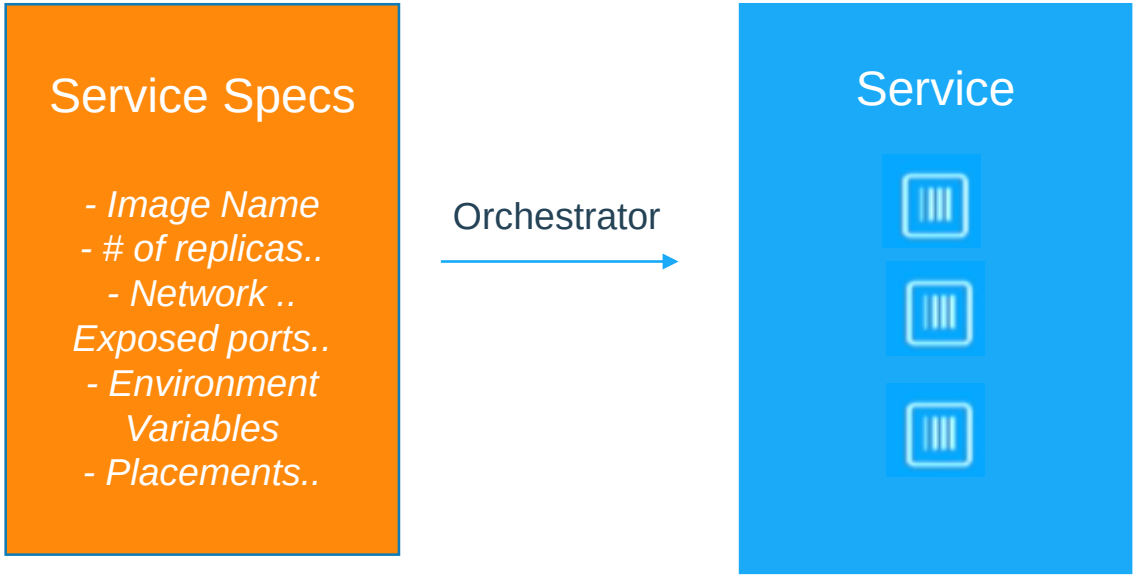
Building Swarm Topology – Hybrid Cluster



http://collabnix.com/building-hybrid-docker-swarm-mode-cluster-on-google-cloud-platform/

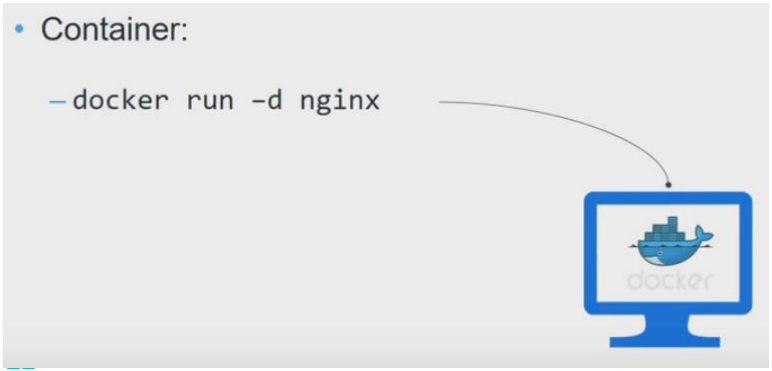
Service Discovery

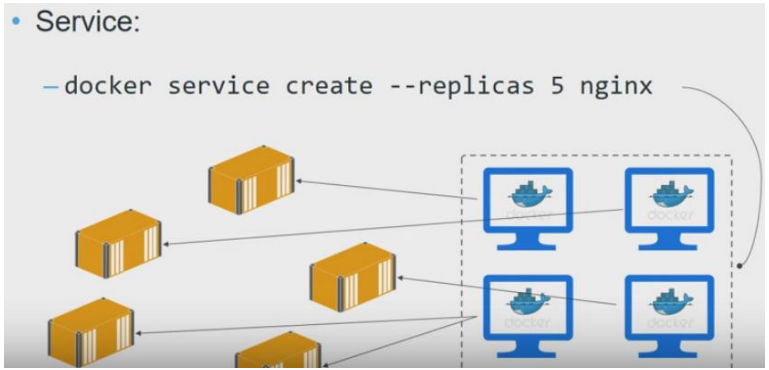
Swarm is built on Services



What is Service?

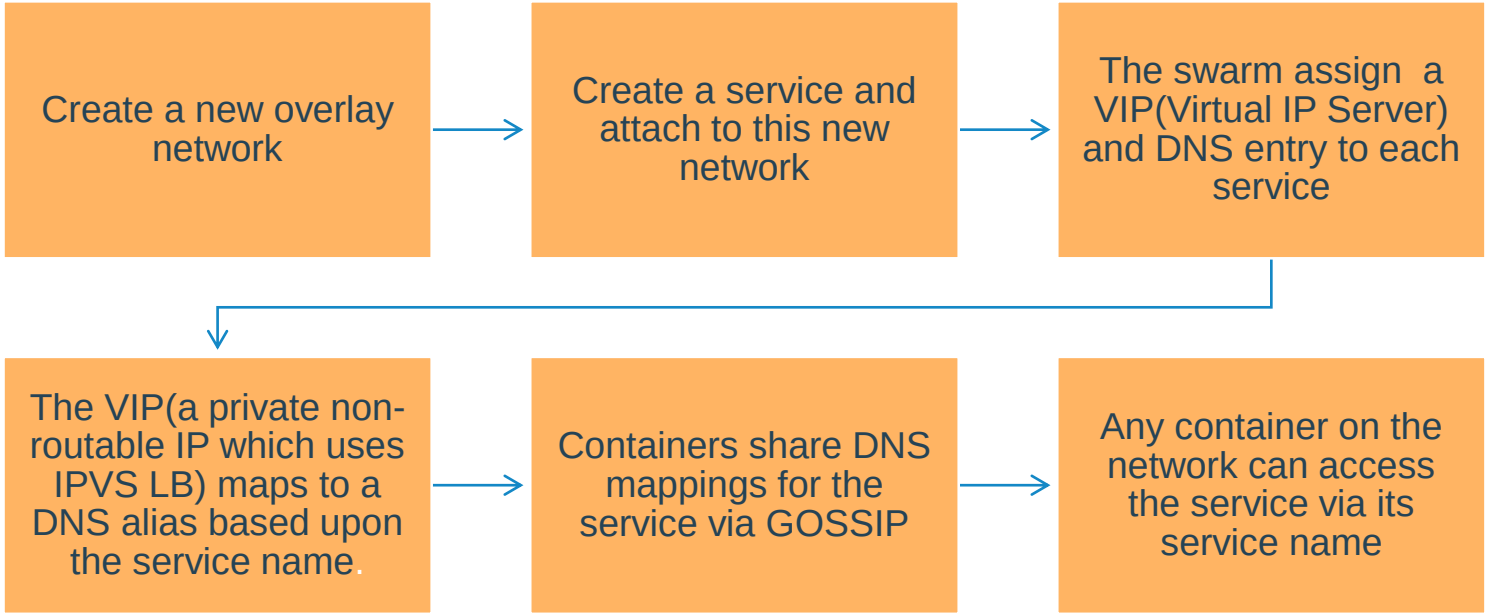
- A definition of tasks to be executed on the worker nodes
- Central structure of swarm system
- An Evolution of `docker run` command
- It manages replicated set of containers
- A task carries a Docker container + commands to run inside the container.





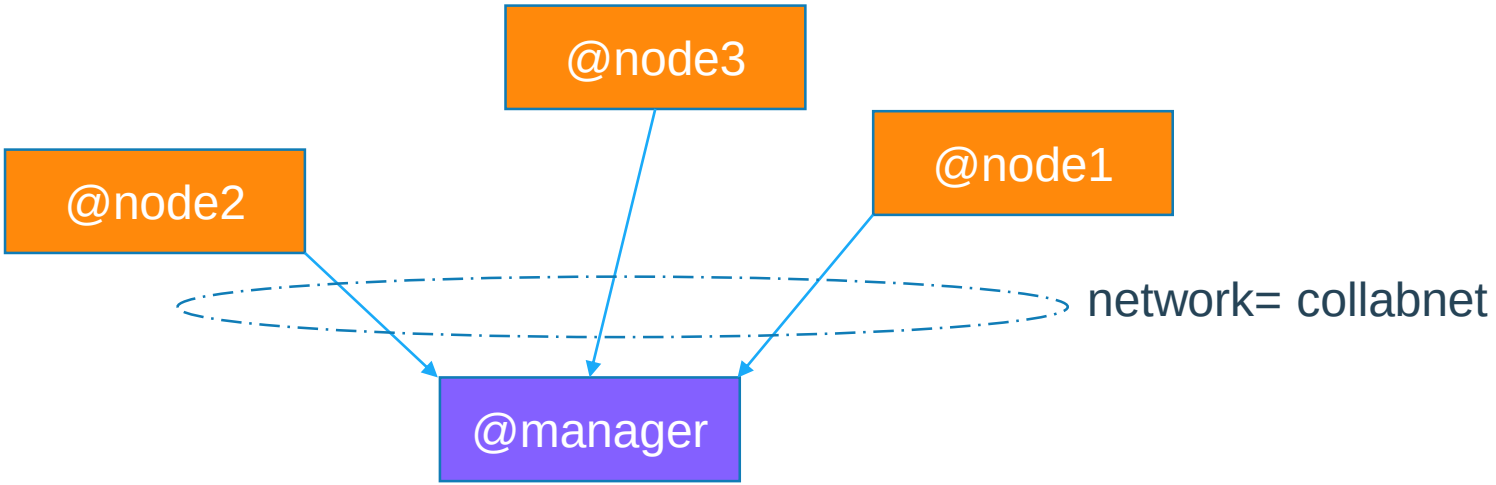
docker

How Service Discovery works in Swarm Mode?



https://collabnix.com/how-service-discovery-works-under-docker-1-12/

Building Our First Swarm Service



$docker network create -d overlay mynetwork

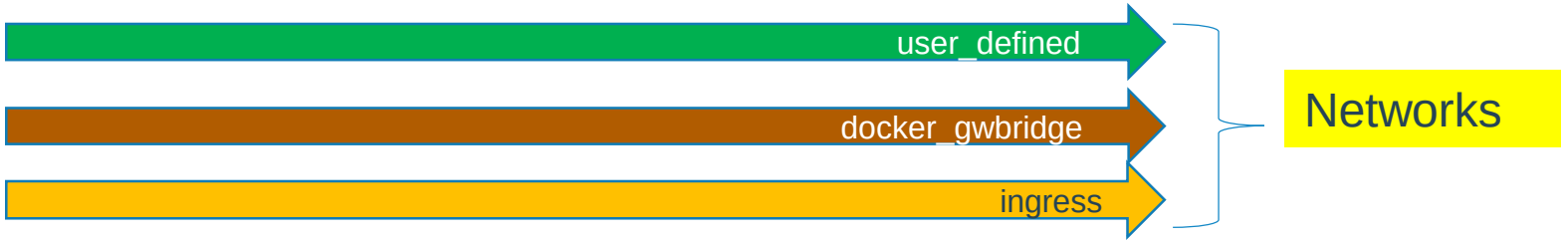
Swarm Cluster Setup

```
master==>sudo docker node ls
ID                            HOSTNAME    STATUS  AVAILABILITY  MANAGER STATUS
1y2tfoe4jo6wjgyqizmmifkx5 *   master1     Ready   Active        Leader
4rzd5krhbo5y2rr4axslrw4jm     test-node2  Ready   Active
4w5oasgfg1pdzngs1rba1wqj6     test-node1  Ready   Active
5t7ftnvm6w44edvkgz9trwo16     test-node3  Ready   Active
```

```
root@master1:~# sudo docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
6f673c1acf3c        bridge              bridge              local
6eaeba0e7c6d        docker_gwbridge     bridge              local
84280982cb88        host                host                local
c4caizphmdpu        ingress             overlay             swarm
ac5edd465975        none                null                local
```

- The default gateway network
- The only network with connectivity to the outside world.(Port 7946 for network discovery

- It is an overlay network on all exposed ports exist.
- Follows a node port model(each service has the same port on every node in the cluster).
- Numbered from 30000 through 32000.
- Used for Routing Mesh(Port 4789 for Ingress)



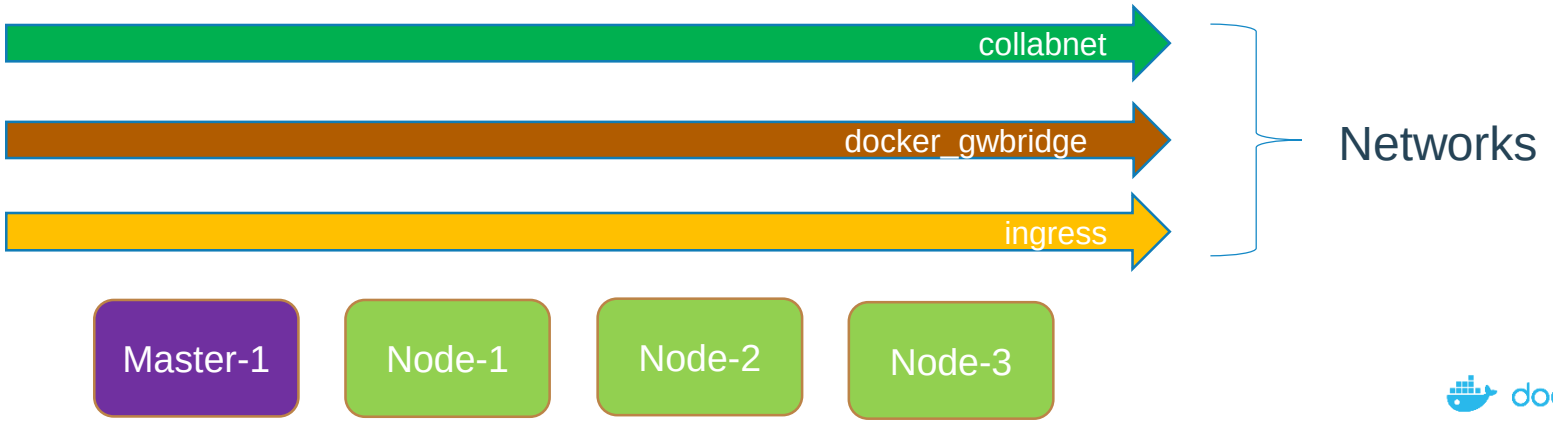
Master-1 | Node-1 | Node-2 | Node-3

docker

Creating a new overlay network

$ docker network create \
    --driver overlay \
    collabnet

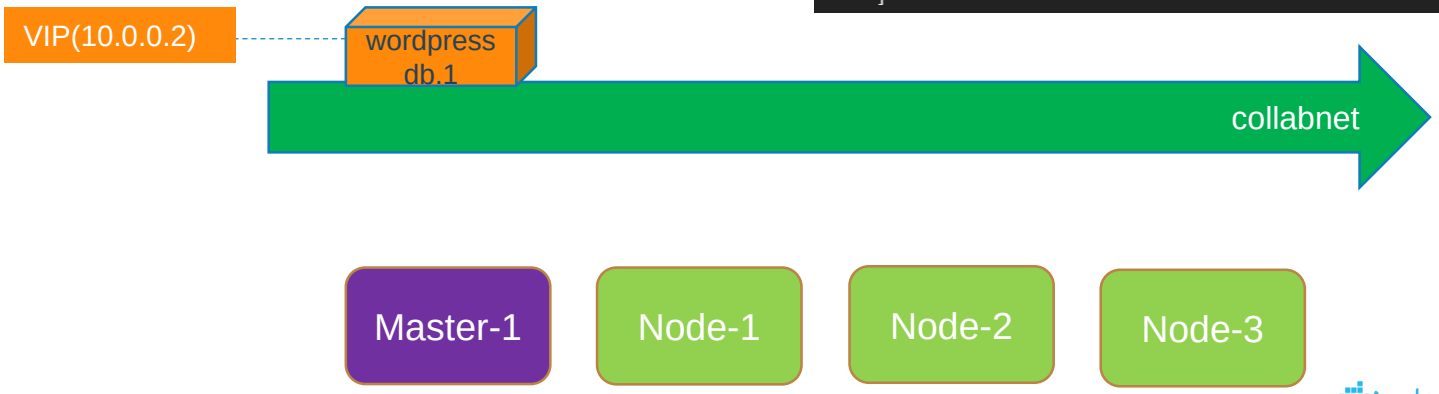
```
master==>sudo docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
1f19d0986d1d        bridge              bridge              local
9eyjm4uv4ynm        collabnet           overlay             swarm
6eaeba0e7c6d        docker_gwbridge     bridge              local
84280982cb88        host                host                local
c4caizphmdpu        ingress             overlay             swarm
ac5edd465975        none                null                local
```



Creating a service “wordpressdb”

```
$ docker service create \
      --replicas 1 \
      --name wordpressdb \
      - -network collabnet \
      -- env MYSQL_ROOT_PASSWORD=collab123 \
      --env MYSQL_DATABASE=wordpress \
      --name wordpressdb  \
      mysql:latest
```

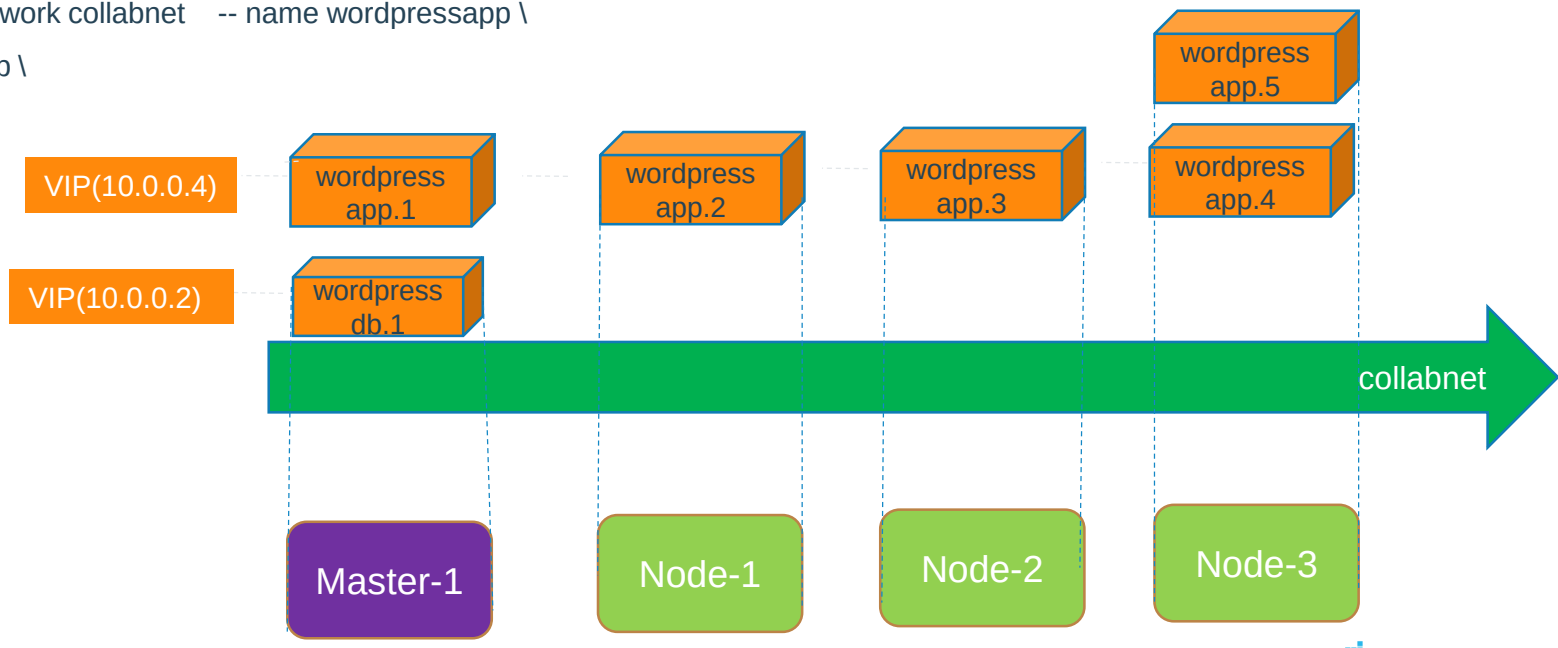
```
},
"Endpoint": {
    "Spec": {
        "Mode": "vip"
    },
    "VirtualIPs": [
        {
            "NetworkID": "9eyjm4uv4ynmz0aubfqxise29",
            "Addr": "10.0.0.2/24"
        }
    ]
```



docker

Creating a service “wordpressapp”

```
$ docker service create \
      --env WORDPRESS_DB_HOST=wordpressdb \
      --env WORDPRESS_DB_PASSWD=collab123 \
      --replicas 5    --network collabnet    -- name wordpressapp \
       --publish 80:80/tcp \
        wordpress:latest
```



docker

Inspecting the services

```
$ docker service inspect \
  --format=='{{json .Endpoint.VirtualIPs}}'  \
  wordpressapp
```

[{"NetworkID":"c4caizphmdpuhm1gjdle8eaal","Addr":"10.255.0.7/16"},

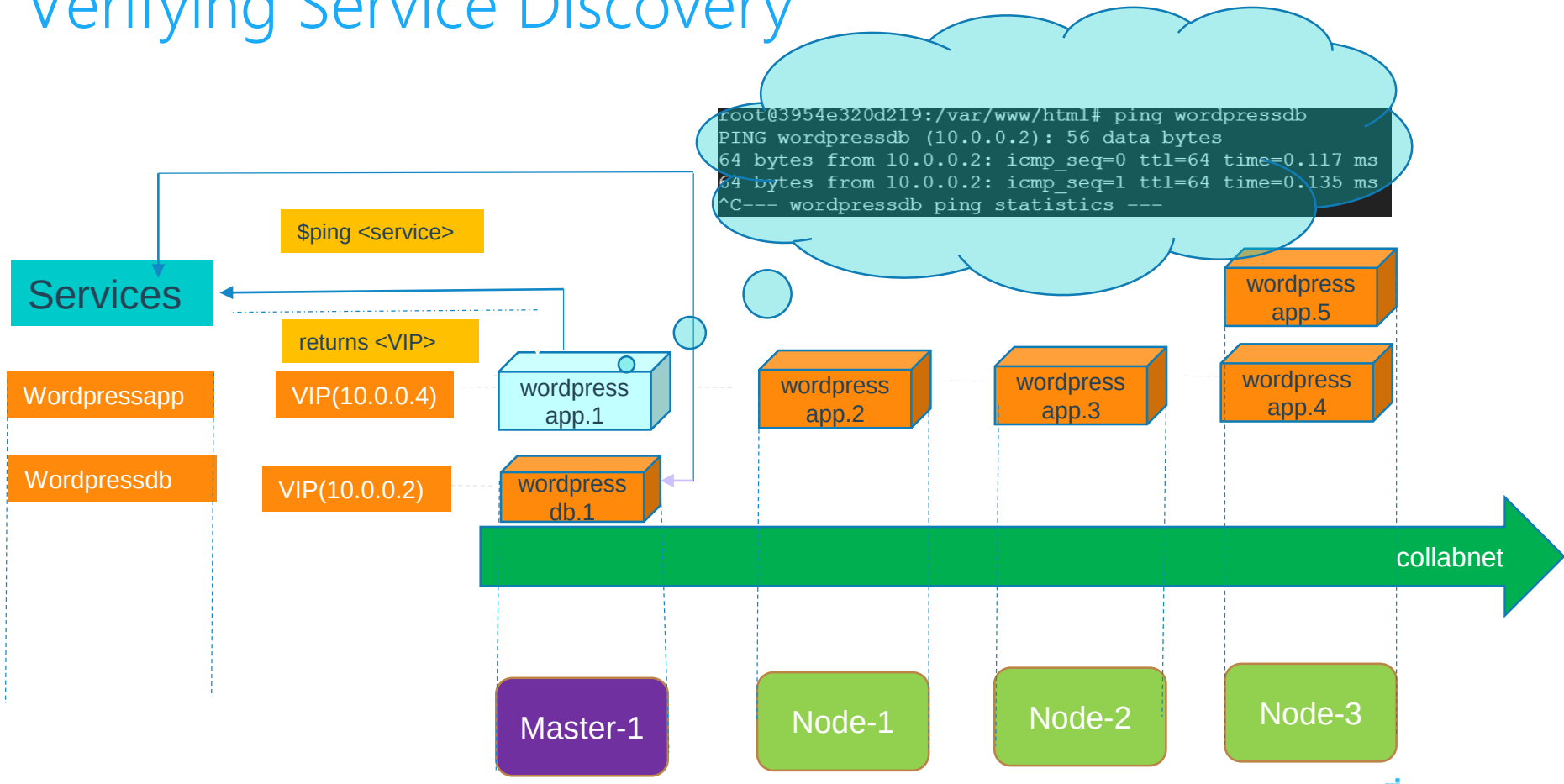
{"NetworkID":"9eyjm4uv4ynmz0aubfqxise29","Addr":"10.0.0.4/24"}]

```
$ docker service inspect \
  --format=='{{json .Endpoint.VirtualIPs}}' \
  wordpressdb
```

[{"NetworkID":"9eyjm4uv4ynmz0aubfqxise29","Addr":"10.0.0.2/24"}]

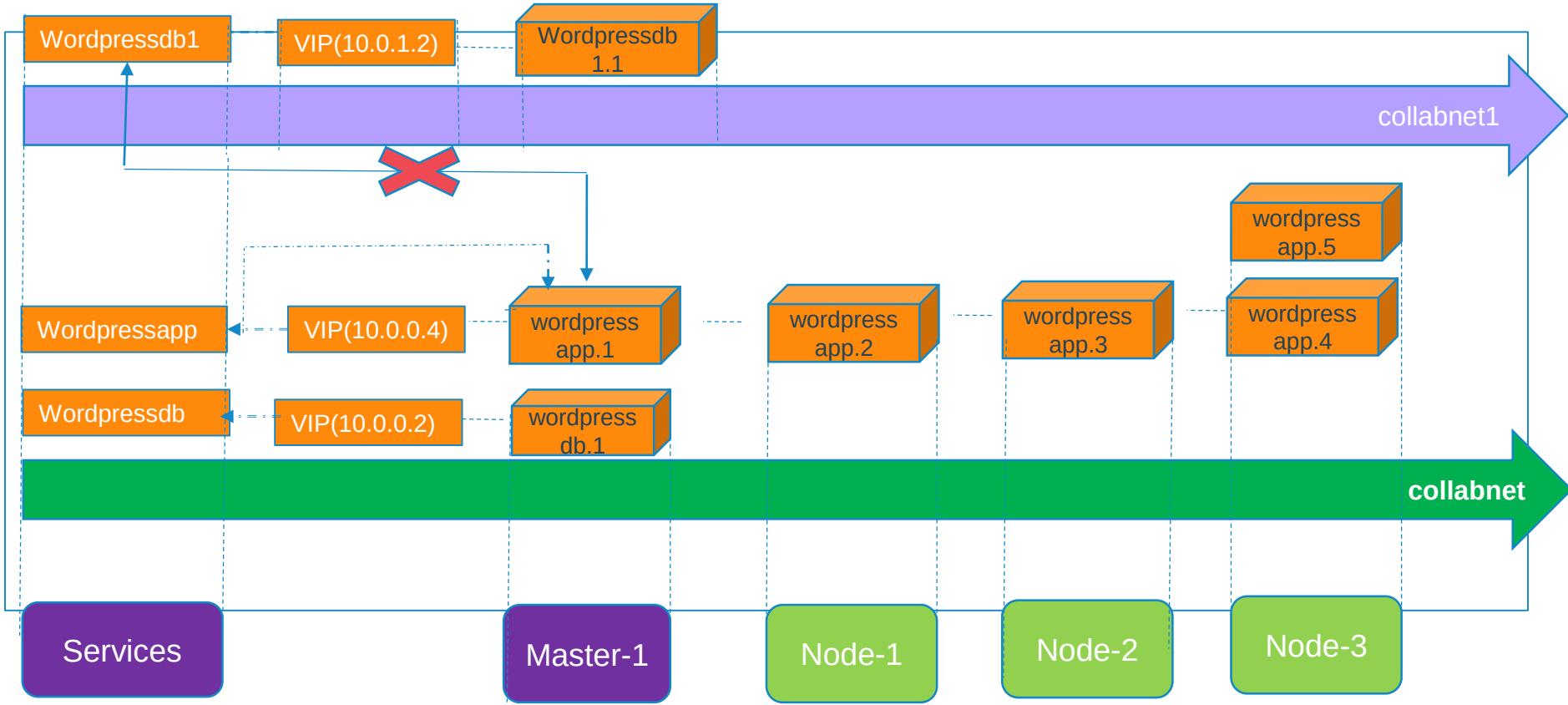
docker

Verifying Service Discovery



docker

Network – A Scope of Service Discoverability



docker

Load Balancing

Load-Balancing

Distributes requests among the healthy nodes.

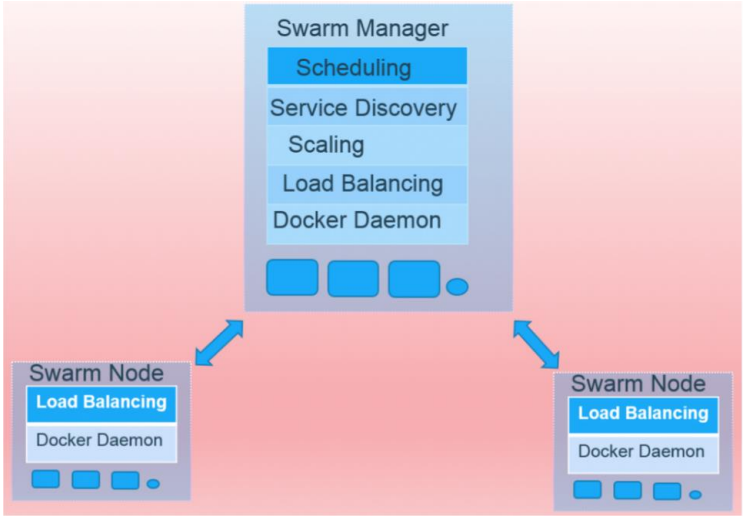
Decentralized, Highly Available – LB instance plumbed into every container instance

Internal Load Balancer – Provided by Embedded DNS

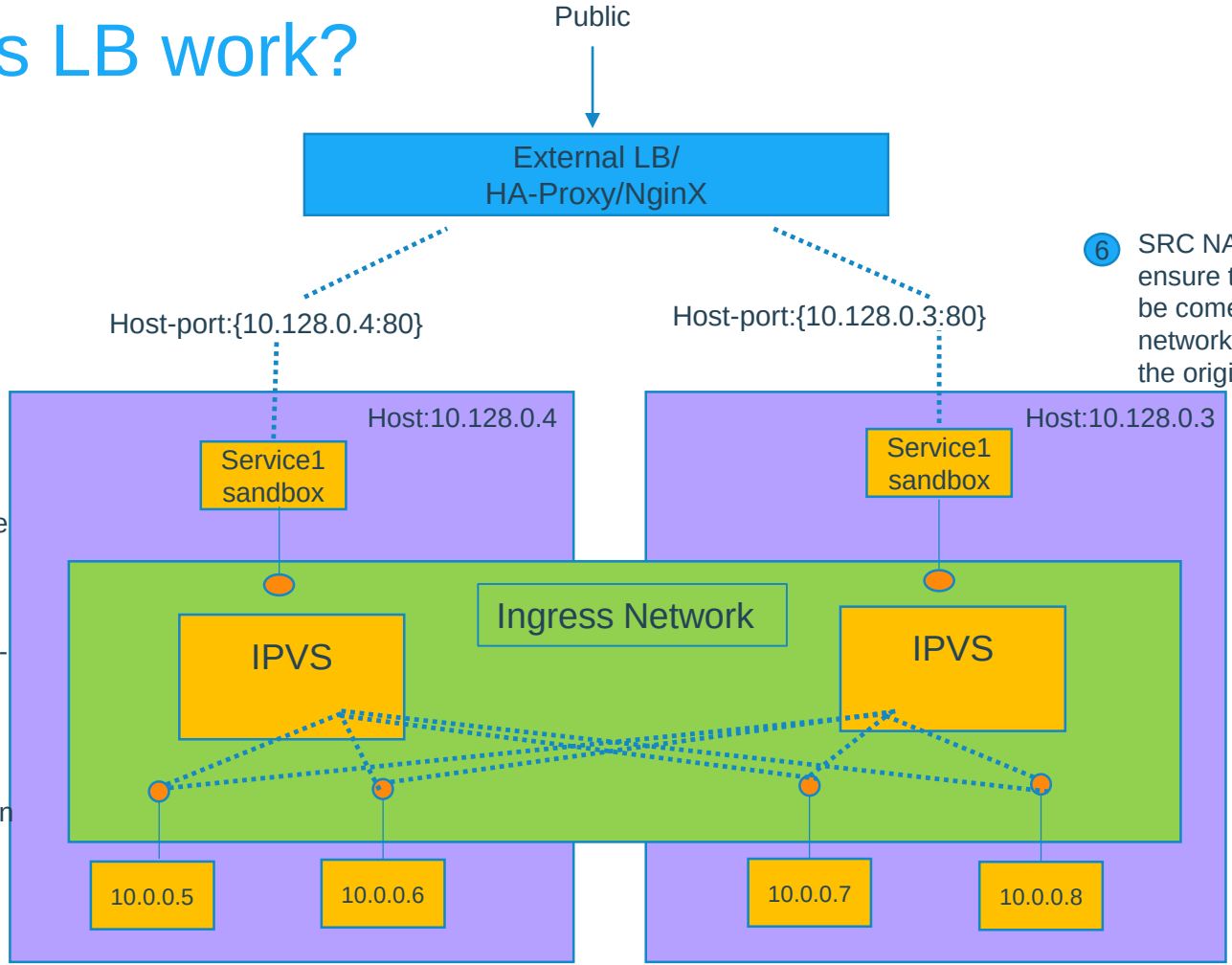
Can be used to discover both service & tasks

VIP based services uses IPVS(IP Virtual Server)

Kernel module (ip_vs) for LB



How does LB work?



1. Client access using :80
2. Plumb the request to sandbox running on 10.128.0.3
3. Packets enters the mangle table, Pre-routing firewall mark of 0x101 => 257
4. Inside the sandbox, the re-routing chain gets created under NAT table.
5. Then ipvsadm uses 257 firewall mark to round robin across the multiple nodes
6. SRC NAT under NAT table ensure that packet has to be come back to Ingress network so as to return in the original format

docker

Accessing the network sandbox

How to find the sandboxID?

```
root@master1:~# docker inspect f0cb | grep -i sandbox
            "SandboxID": "5fcd63cde83887d9033cac9fecd1f2c66de6b71648d660a6f80355be08aebb9f",
            "SandboxKey": "/var/run/docker/netns/5fcd63cde838",
root@master1:~# docker inspect d38c5 | grep -i sandbox
            "SandboxID": "36c249f1ef1dfa1fed03b85126588d4e3922dd5cb426d0c97ff800c5418e0dff",
            "SandboxKey": "/var/run/docker/netns/36c249f1ef1d",
root@master1:~# ls /var/run/docker/netns/
1-9eyjm4uv4y  1-c4caizphmd  36c249f1ef1d  5fcd63cde838  bf4eb2c9e6d7
```

Where's sandbox located?

```
root@master1:~# ls /var/run/docker/netns
1-9eyjm4uv4y  1-c4caizphmd  36c249f1ef1d  5fcd63cde838  bf4eb2c9e6d7
root@master1:~# nsenter --net=/var/run/docker/netns/5fcd63cde838 sh
```

Network namespace managed by overlay network driver(creating a bridge, terminating VXLAN tunnel etc.

Inspecting the sandbox

```
# iptables -t mangle -nvL
Chain PREROUTING (policy ACCEPT 291 packets, 45678 bytes)
 pkts bytes target     prot opt in     out     source               destination
  164 10845 MARK       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpt:80 MARK set 0x101

Chain INPUT (policy ACCEPT 164 packets, 10958 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 127 packets, 34720 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 164 packets, 10870 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 MARK       all  --  *      *       0.0.0.0/0            10.255.0.7          MARK set 0x101

Chain POSTROUTING (policy ACCEPT 291 packets, 45590 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

```
root@master1:~# nsenter --net=/var/run/docker/netns/bf4eb2c9e6d7 sh
# iptables -t nat -nvL
Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
   45  2160 REDIRECT   tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp dpt:80 redir ports 80

Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DOCKER_OUTPUT  all  --  *      *       0.0.0.0/0            127.0.0.11

Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DOCKER_POSTROUTING  all  --  *      *       0.0.0.0/0            127.0.0.11
   45  2160 SNAT       all  --  *      *       0.0.0.0/0            0.0.0.0/0           ipvs to:10.255.0.3

Chain DOCKER_OUTPUT (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DNAT       tcp  --  *      *       0.0.0.0/0            127.0.0.11          tcp dpt:53 to:127.0.0.11:33915
    0     0 DNAT       udp  --  *      *       0.0.0.0/0            127.0.0.11          udp dpt:53 to:127.0.0.11:56454

Chain DOCKER_POSTROUTING (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 SNAT       tcp  --  *      *       127.0.0.11           0.0.0.0/0           tcp spt:33915 to::53
    0     0 SNAT       udp  --  *      *       127.0.0.11           0.0.0.0/0           udp spt:56454 to::53
```

docker

Routing Mesh

Routing Mesh is NOT Load-Balancer

Routing Mesh makes use of LB aspects

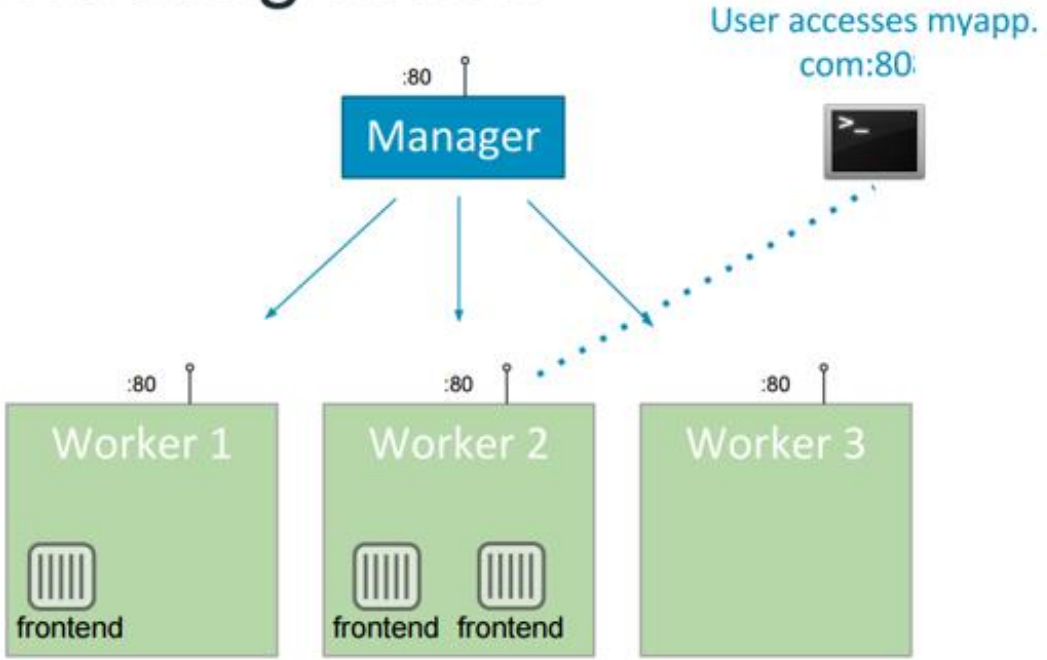
It provides global publish port for a given service

Built-in routing mesh for edge routing

Worker nodes themselves participate in ingress routing mesh

Port management at global Swarm Cluster level.

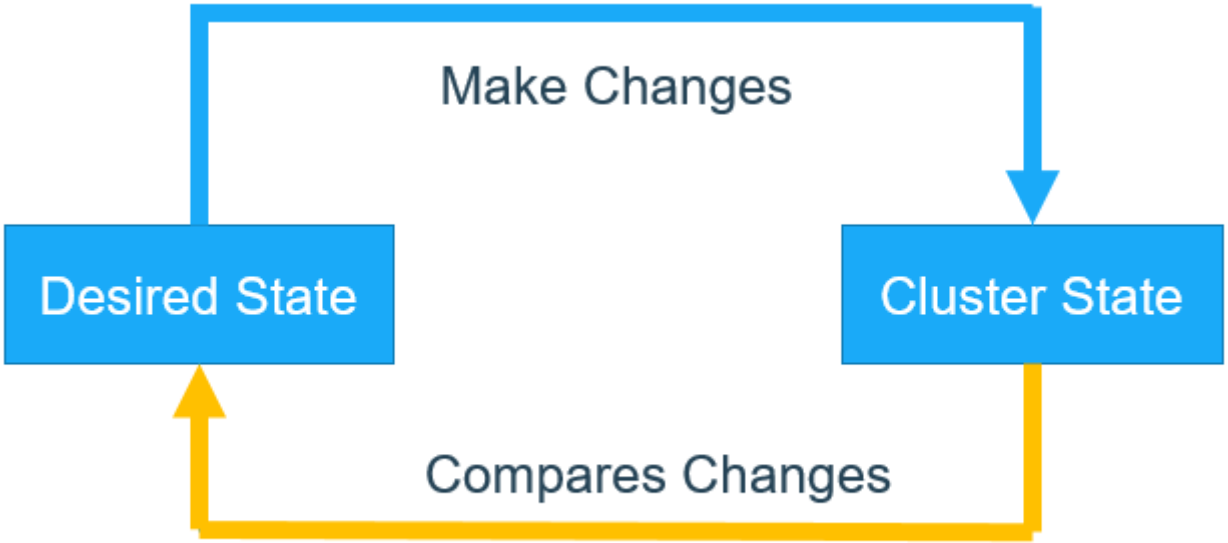
Routing Mesh



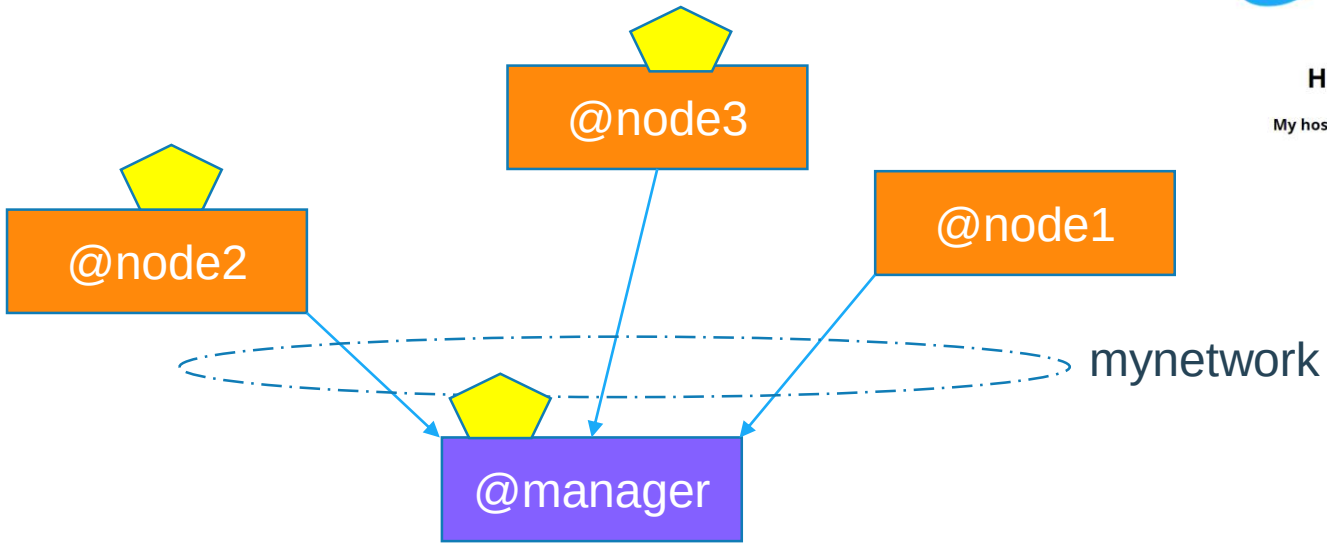
- Operator reserves a swarm-wide ingress port (80) for `myapp`
- Every node listens on 80
- Container-aware routing mesh can transparently reroute traffic from Worker3 to a node that is running container
- Built in load balancing into the Engine
- DNS-based service discovery

```
$ docker service create --replicas 3 --name frontend --network mynet
  --publish 80:80/tcp frontend_image:latest
```

Desired State Reconciliation

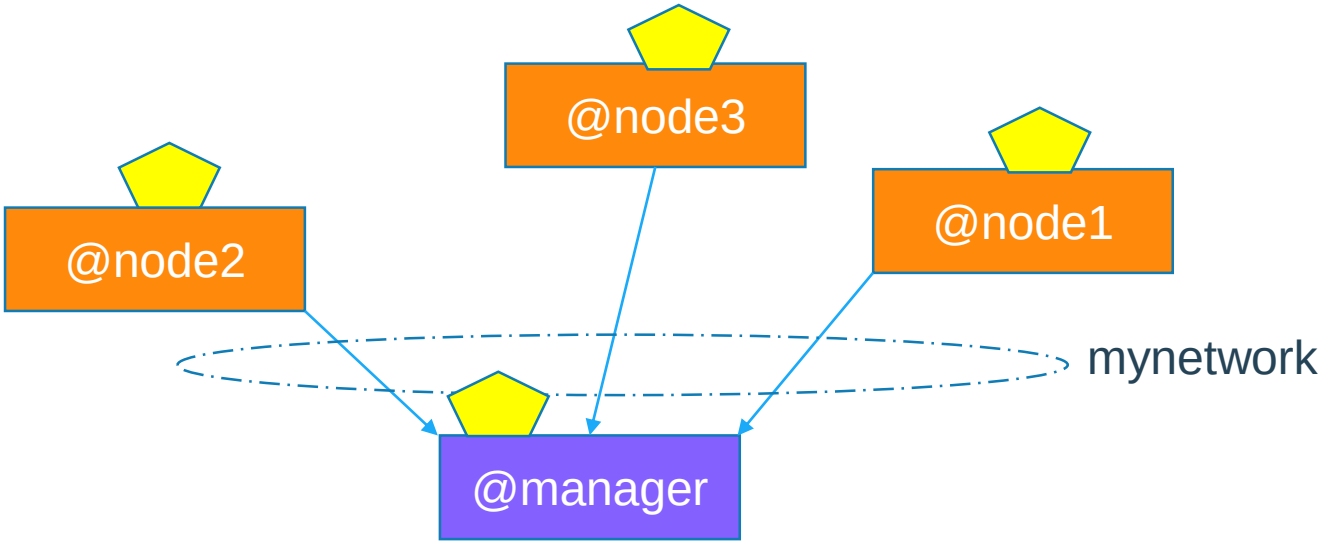


Building Our First Swarm Service



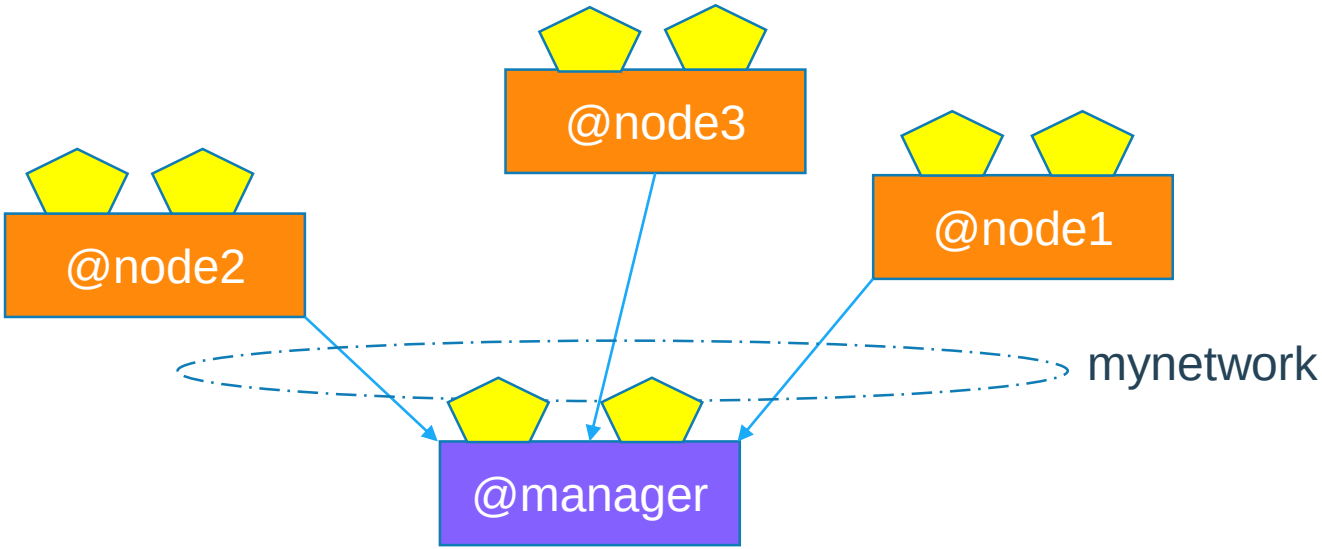
$docker network create -d overlay mynetwork
$docker service create --name mycloud --replicas 3 --network mynetwork --publish 80:80/tcp dockercloud/hello-world

Swarm Services – [Desired State ≠ Actual State]



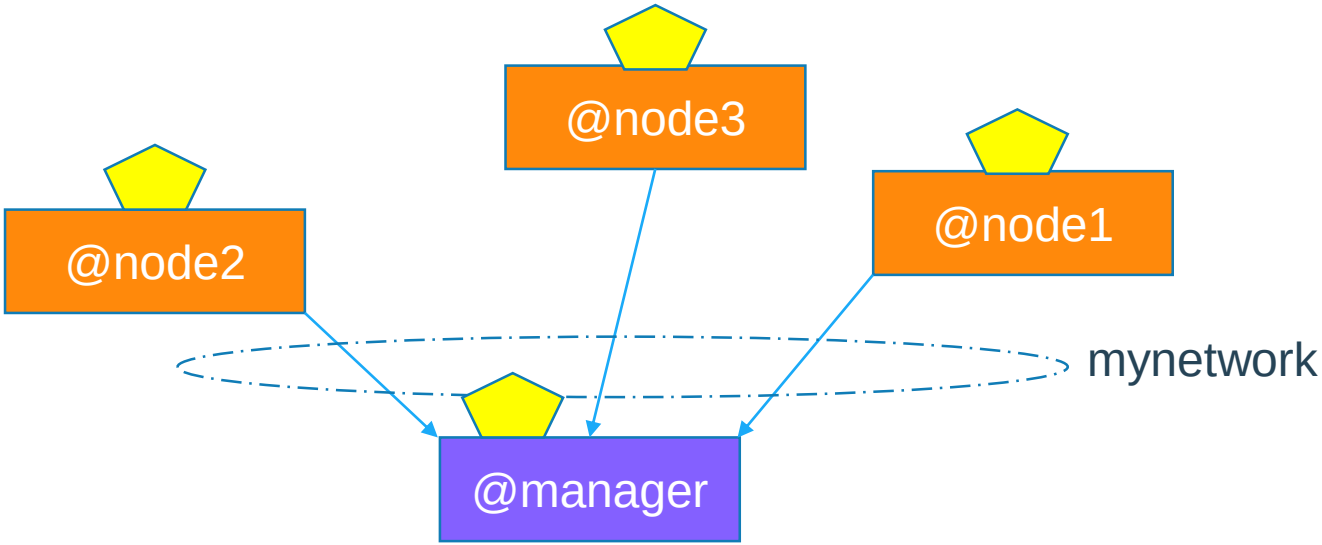
$docker service scale mycloud=8

Swarm Services – [Desired State = Actual State]



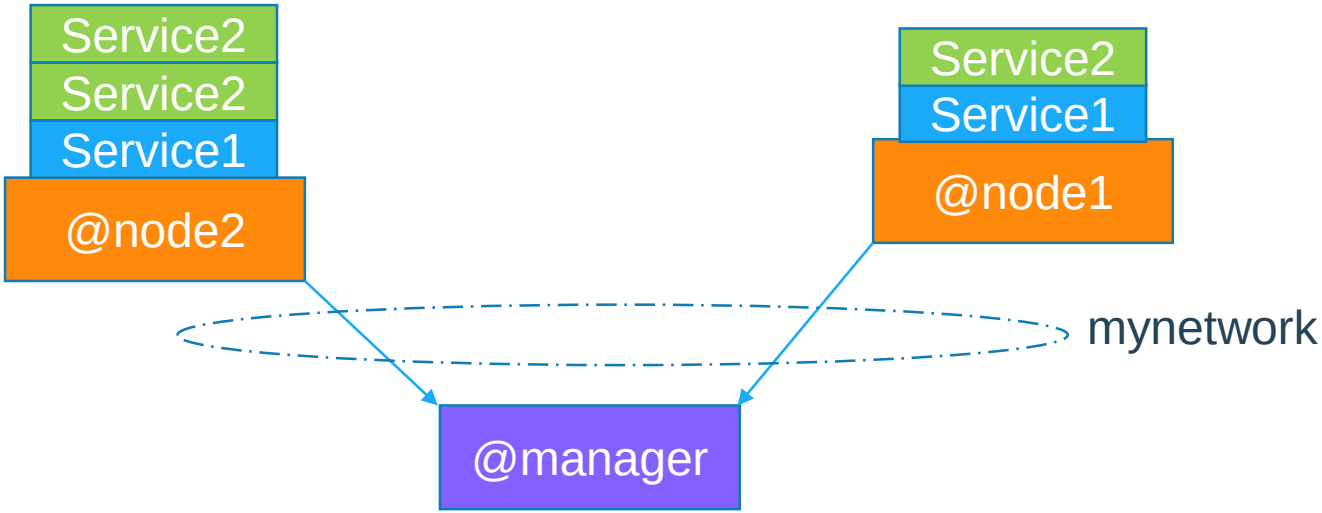
$docker service scale mycloud=8

Swarm Mode – Global Services



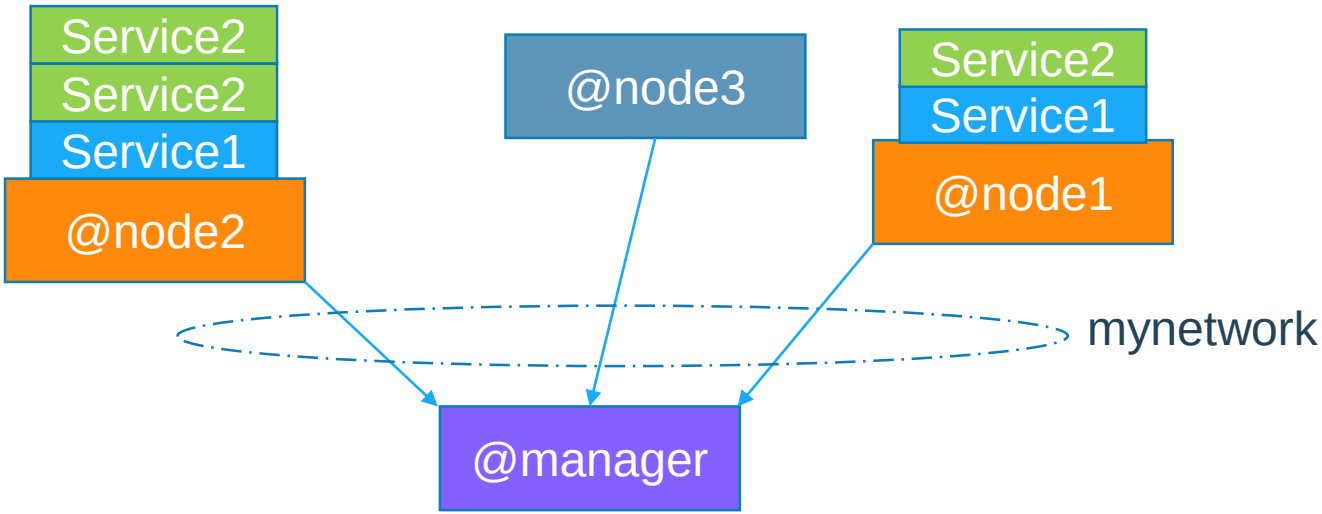
$docker service create –mode=global –name mycloud dockercloud/hello-world

Swarm Mode – High Availability Scheduling



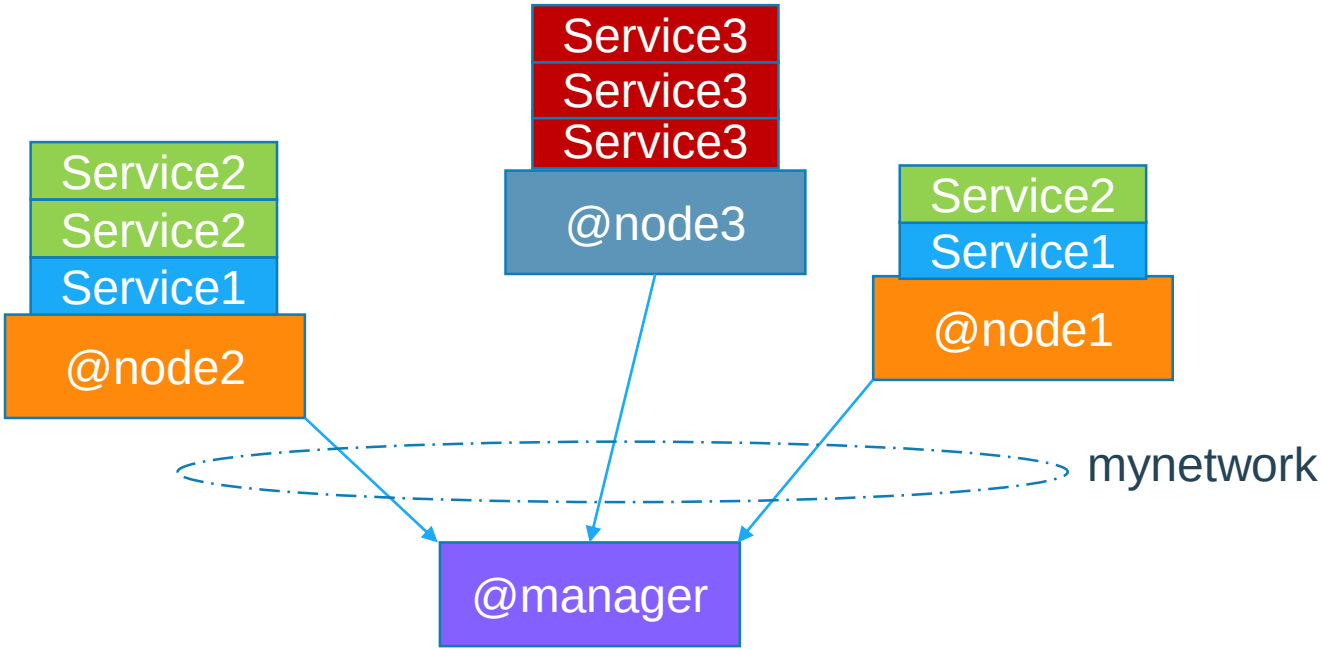
Prioritizing spreading out the containers instead of equalizing the number of containers per node

Swarm Mode – High Availability Scheduling



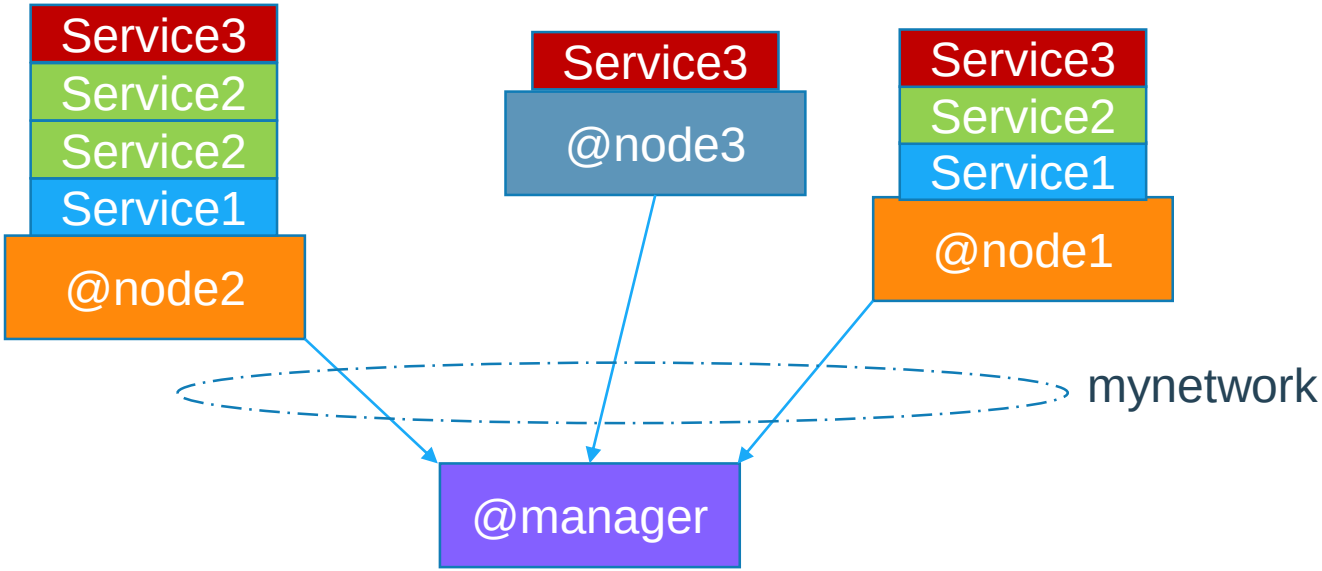
Adding a new Node - @node3

Swarm Mode – High Availability Scheduling



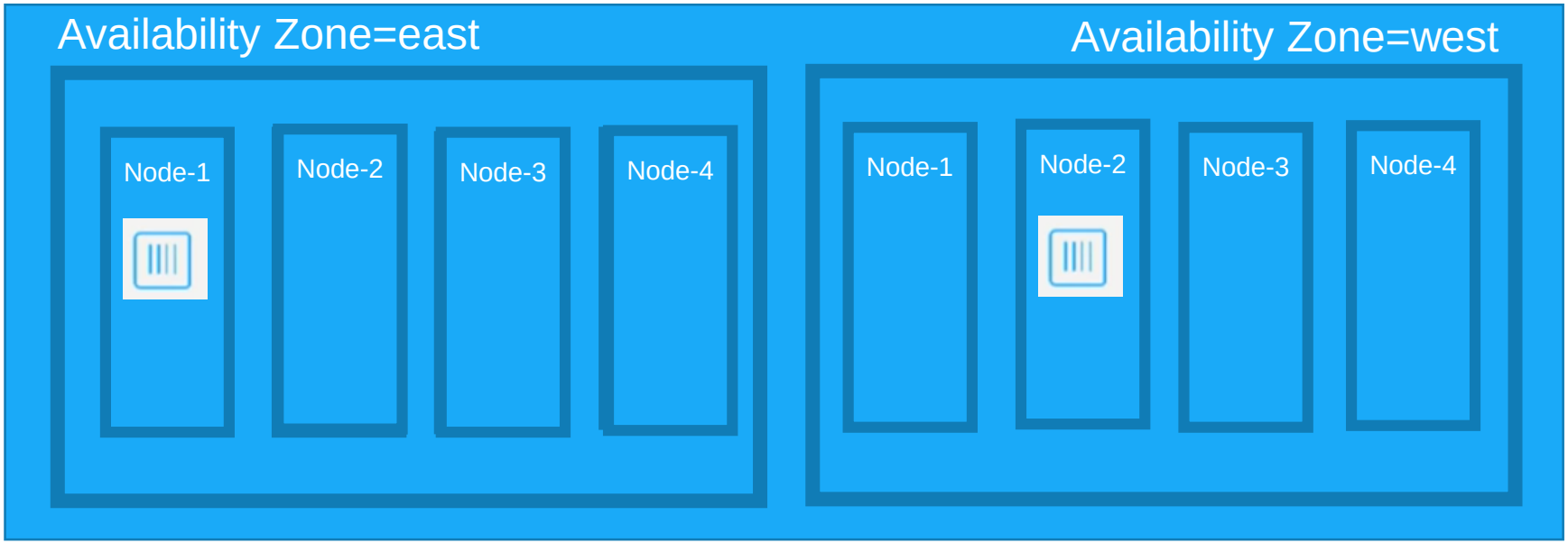
Prioritizing spreading out the containers instead of equalizing the number of containers per node

Swarm Mode – High Availability Scheduling



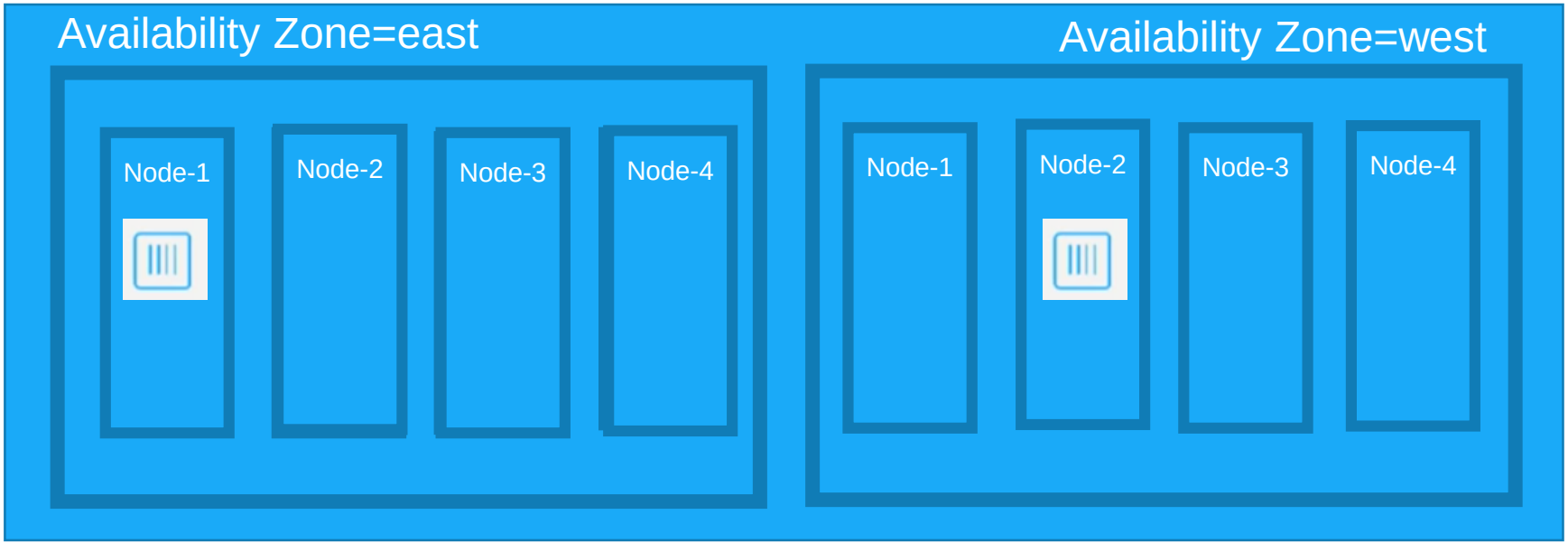
Prioritizing spreading out the containers instead of equalizing the number of containers per node

Swarm Mode – Topology Aware Scheduling



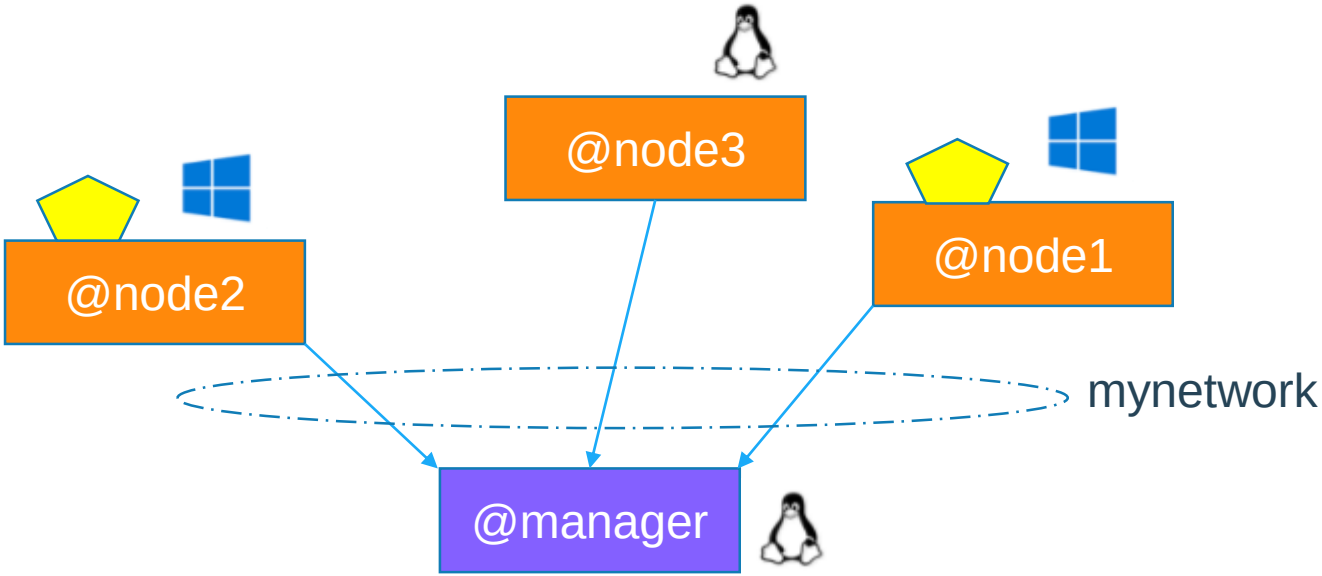
$docker node update --label-add datacenter=east node-1

Topology Aware Scheduling – How to use it?

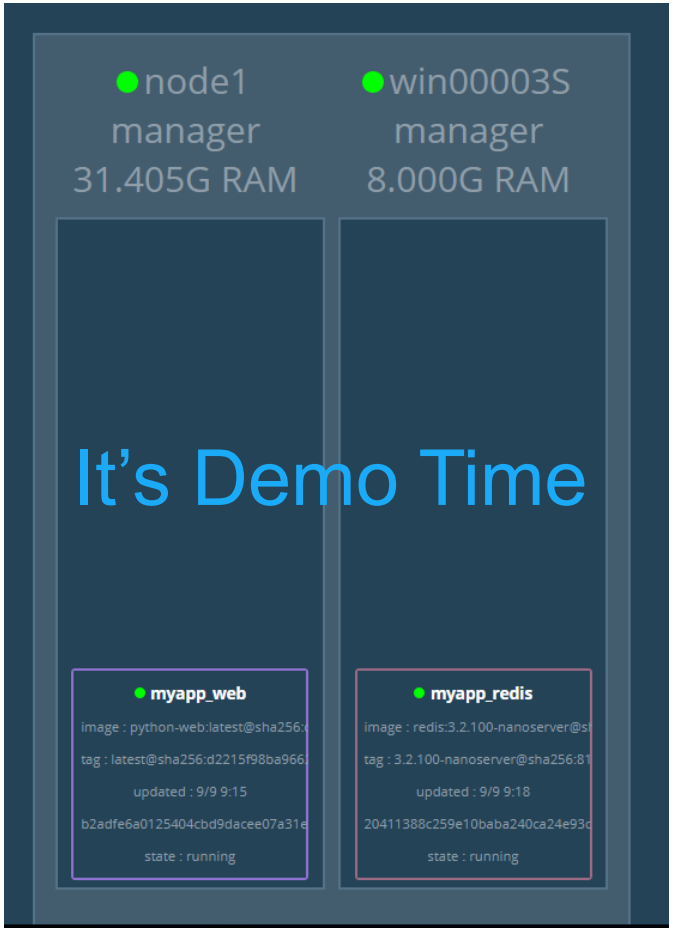


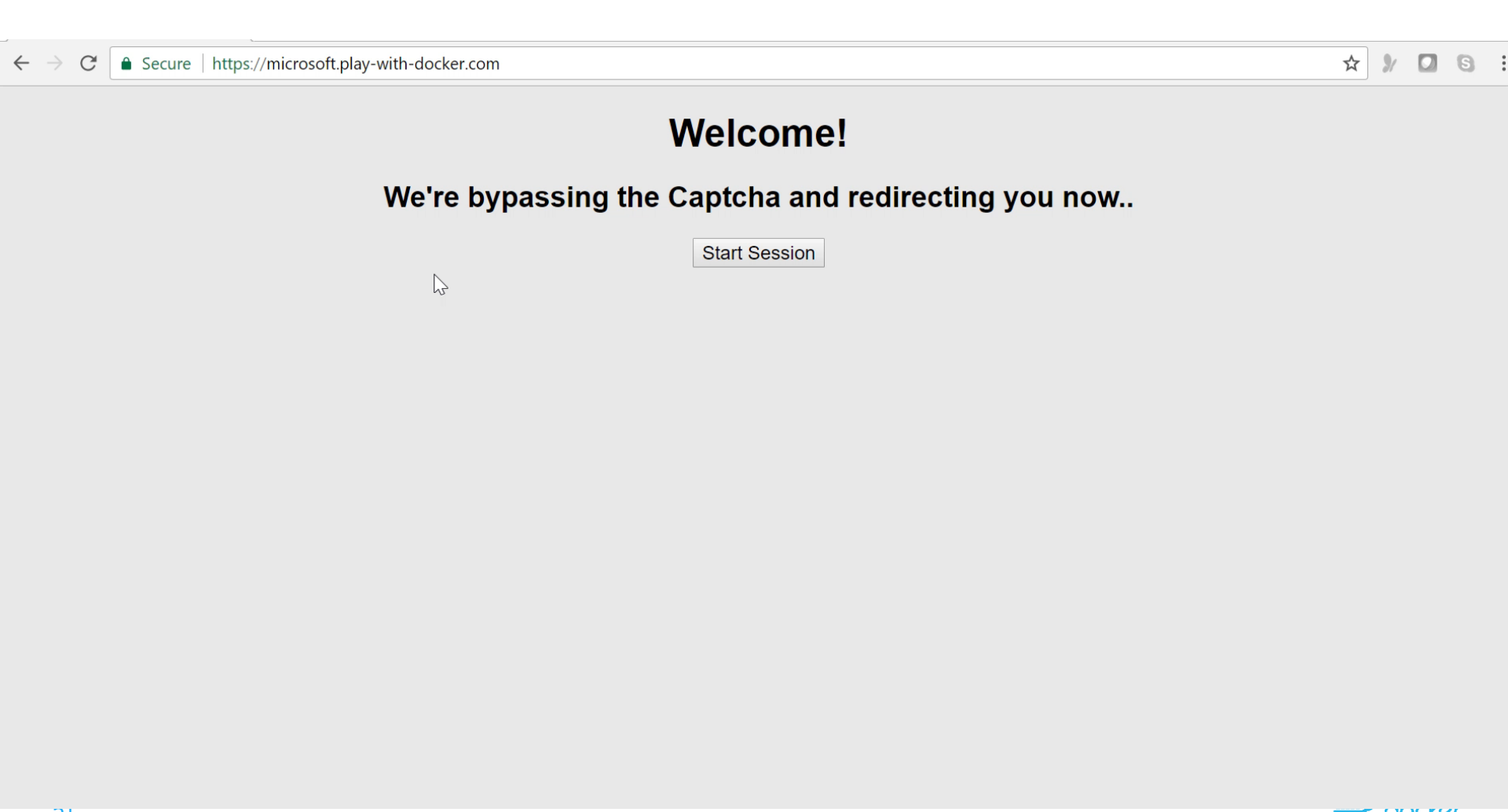

```
$docker service create --replicas 2 --name wordpressdb1 \
--network collabnet --placement-pref “spread=node.labels.datacenter” \
 --env MYSQL_ROOT_PASSWORD=collab123 \
--env MYSQL_DATABASE=wordpress mysql:latest
```

Swarm Mode – Placement Constraints




```
$docker service create --network collabnet \
    --endpoint-mode dnsrr \
    --constraint ‘node.platform.os == windows’ \
    --env ACCEPT_EULA=Y    --env-file db-credentials.env \
    --name db microsoft/mssql-server-windows
```


node1
manager
31.405G RAM
win00003S
manager
8.000G RAM
It’s Demo Time
myapp_web
image : python-web:latest@sha256:
tag : latest@sha256:d2215f98ba966
updated : 9/9 9:15
b2adfe6a0125404cbd9dacee07a31e
state : running
myapp_redis
image : redis:3.2.100-nanoserver@s
tag : 3.2.100-nanoserver@sha256:8
updated : 9/9 9:18
20411388c259e10baba240ca24e93c
state : running


Secure
https://microsoft.play-with-docker.com
Welcome!
We're bypassing the Captcha and redirecting you now..
Start Session


docker
con 17 EU
Bella Center, Copenhagen
16-19 October
dockercon17

docker