CLOUD
COMPUTING
APPLICATIONS

Docker Ingress Network and Routing Mesh

Prof. Reza Farivar

Swarm Overlay Networks

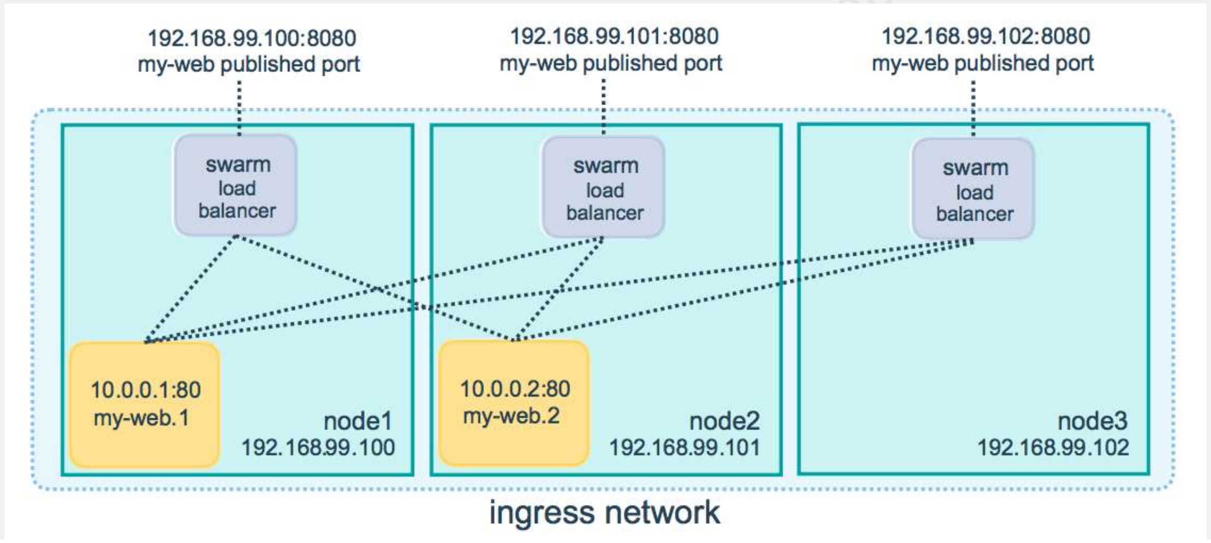
- Recall bridge networks on one host:
  - Containers connected to the same user-defined bridge network effectively expose all ports to each other.
  - For a port to be accessible to containers or non-Docker hosts on different networks, that port must be *published* using the -p or --publish flag.
- On a multi-host Docker Swarm
  - Swarm services connected to the same overlay network effectively expose all ports to each other
  - For a port to be accessible outside of the service, that port must be *published* using the -p or --publish
- By default, swarm services which publish ports do so using the ***routing mesh***

Docker Swarm Routing Mesh

- By default, swarm services which publish ports do so using the routing mesh
- When you connect to a published port on any swarm node **(whether it is running a given service or not)**, you are redirected to a worker which is running that service, transparently
- Effectively, Docker acts as a load balancer for your swarm services.
- Services using the routing mesh are running in *virtual IP (VIP) mode*.

Published ports

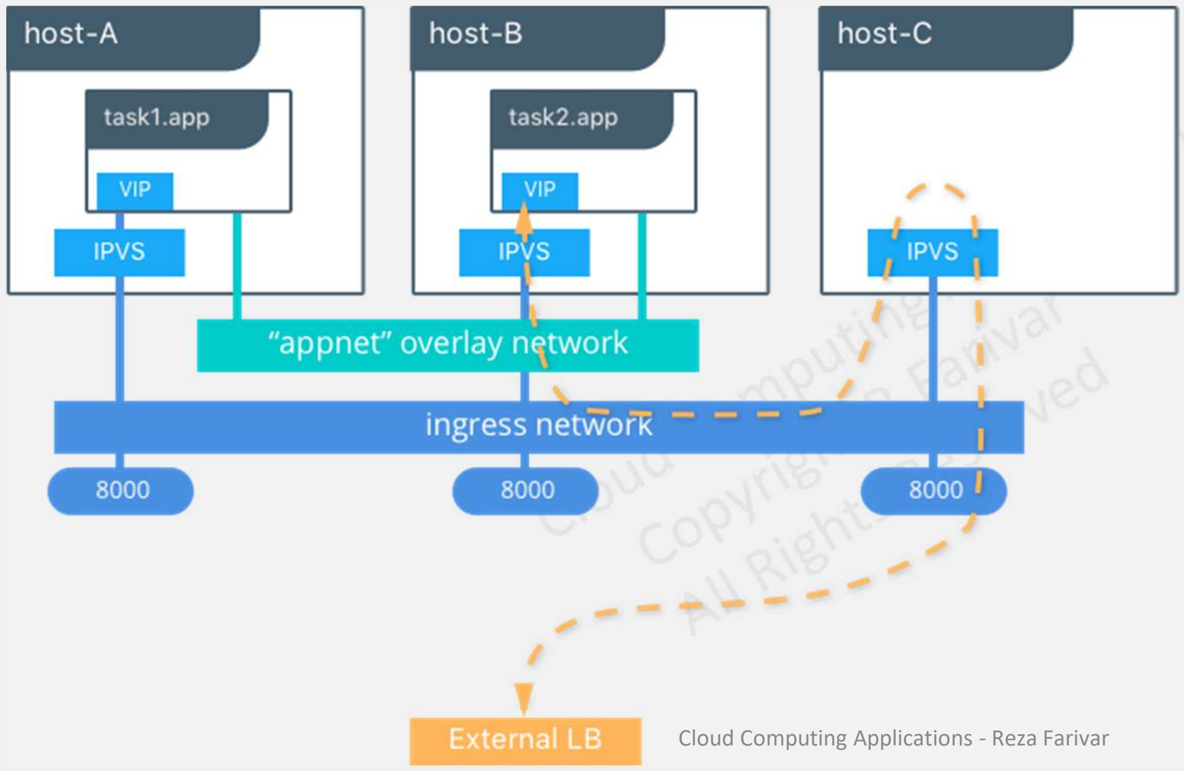
- When you create a swarm service, you can publish that service's ports to hosts outside the swarm
  - Routing mesh



ingress Overlay Network

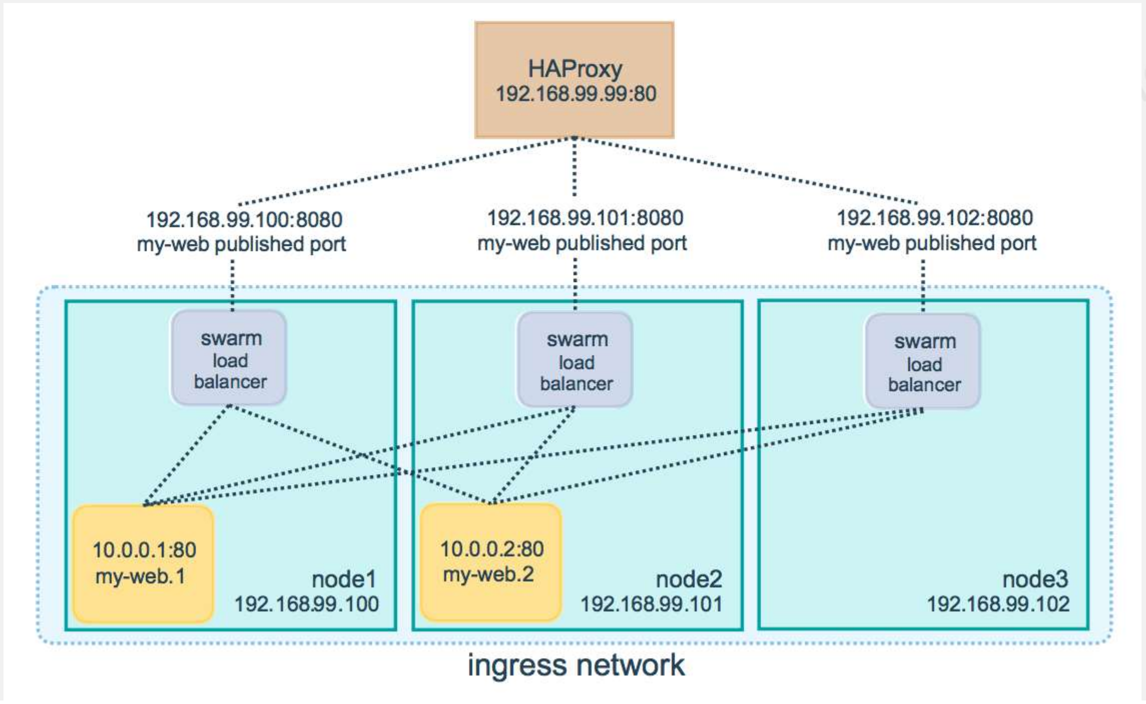
- The **ingress** network plays an important role in enabling published ports over the swarm
  - When you create a swarm service and do not connect it to a user-defined overlay network, it connects to the ingress network by default
- For a port to be accessible ***outside*** of the swarm service, it must be *published* using the `-p` or `--publish` flag on `docker service create` or `docker service update`
- Map TCP port 80 on the service to TCP port 8080 on the routing mesh, and map UDP port 80 on the service to UDP port 8080 on the routing mesh.
  - `-p 8080:80/tcp -p 8080:80/udp`
  - `-p published=8080,target=80,protocol=tcp -p published=8080,target=80,protocol=udp`

External Load Balancing in Swarm Services



External Load Balancer

- Using the routing mesh



Example HAProxy Config

```
global
        log /dev/log    local0
        log /dev/log    local1 notice
...snip...

# Configure HAProxy to listen on port 80
frontend http_front
   bind *:80
   stats uri /haproxy?stats
   default_backend http_back

# Configure HAProxy to route requests to swarm nodes on port 8080
backend http_back
   balance roundrobin
   server node1 192.168.99.100:8080 check
   server node2 192.168.99.101:8080 check
   server node3 192.168.99.102:8080 check
```

External Load Balancing w/o Routing Mesh

- set `--endpoint-mode` to `dnsrr`
  - the default value is `vip`
- No more a single virtual IP
- Docker sets up DNS entries for the service such that a DNS query for the service name returns ***a list of IP addresses***
  - Client connects directly to one of these
- You are responsible for providing the list of IP addresses and ports to your load balancer

Routing Mesh

- When you publish a service port, the swarm makes the service accessible at the target port on every node
  - regardless of whether there is a task for the service running on that node or not.
- This is less complex and is the right choice for many types of services

Host Mode

- You can publish a service task's port directly on the swarm node where that service is running.
  - Bypasses the routing mesh
  - Provides the maximum flexibility, including the ability to develop custom routing framework
- However, you are responsible for keeping track of where each task is running and routing requests to the tasks, and load-balancing across the nodes.
- use the `mode=host` option to the `--publish` flag