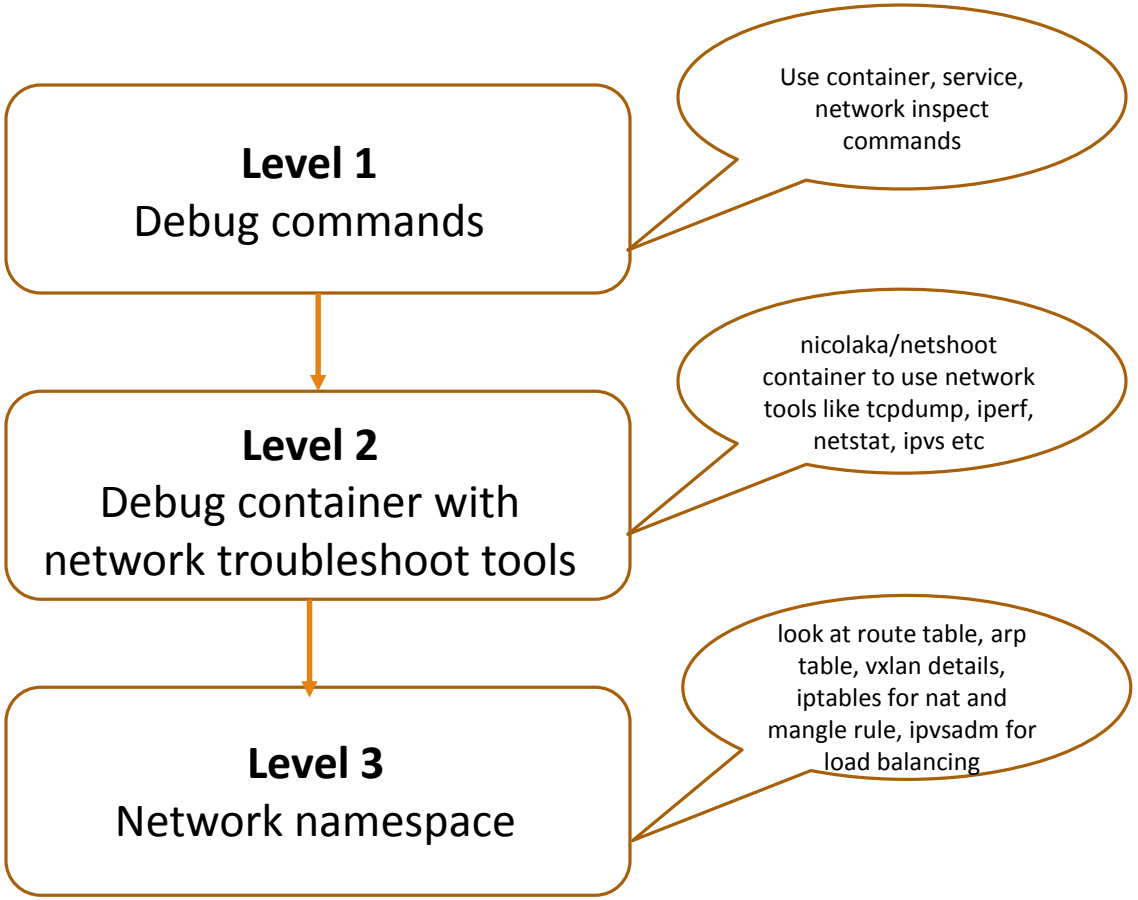
DOCKER NETWORKING TIPS

Troubleshooting

Presenter's Name: Sreenivas Makam

Associated Youtube video: https://youtu.be/d-sgEwmvBko

Troubleshooting levels



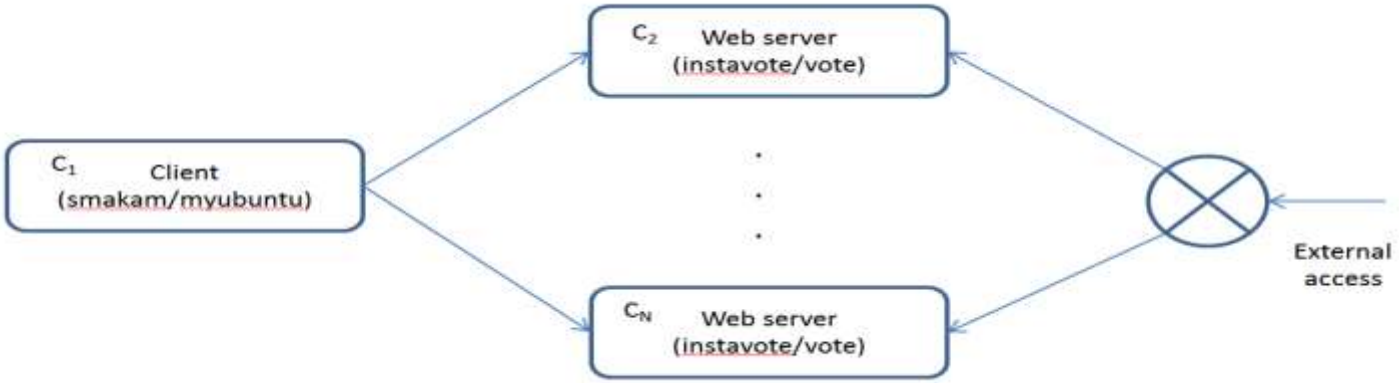
Swarm Networking - Sample application detail

- The application will be deployed in 2 node Swarm cluster.
- "client" service has 1 client container task. "vote" service has multiple vote container tasks. Client service is used to access multi-container voting service. This application is deployed in a multi-node Swarm cluster.
- "vote" services can be accessed from "client" service as well as from outside the swarm cluster.

docker network create -d overlay overlay1

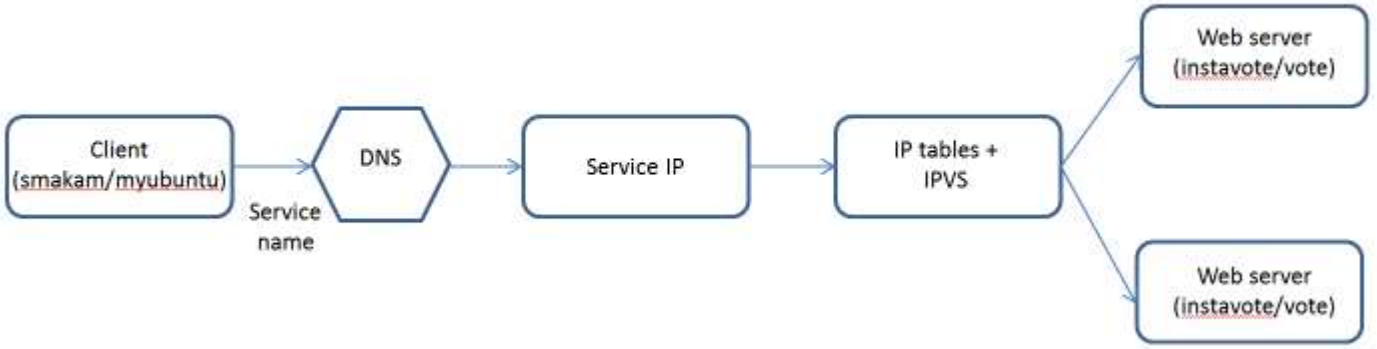
docker service create --replicas 1 --name client --network overlay1 smakam/myubuntu:v4 sleep infinity

docker service create --name vote --network overlay1 --mode replicated --replicas 2 --publish mode=ingress,target=80,published=8080 instavote/vote

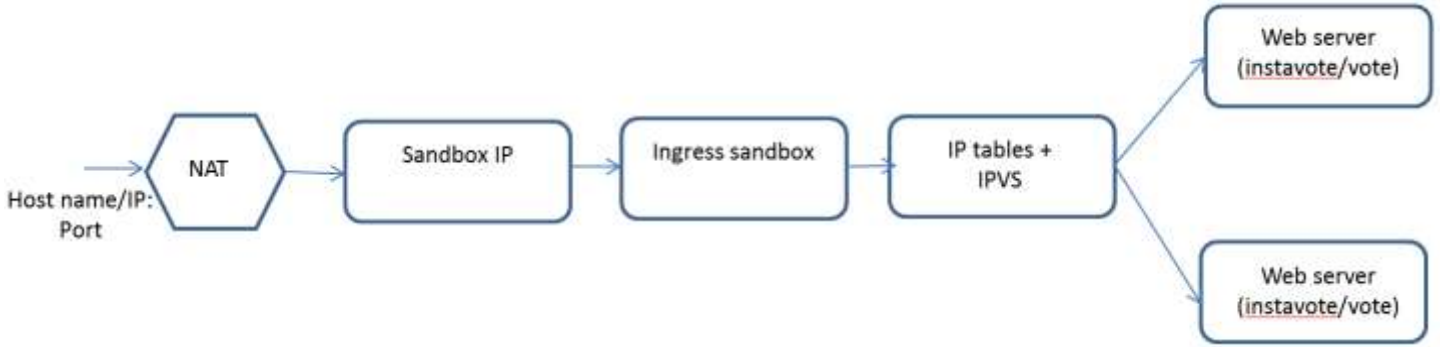


Swarm Networking - Application access flow

“Client” service accessing “vote” service using “overlay” network

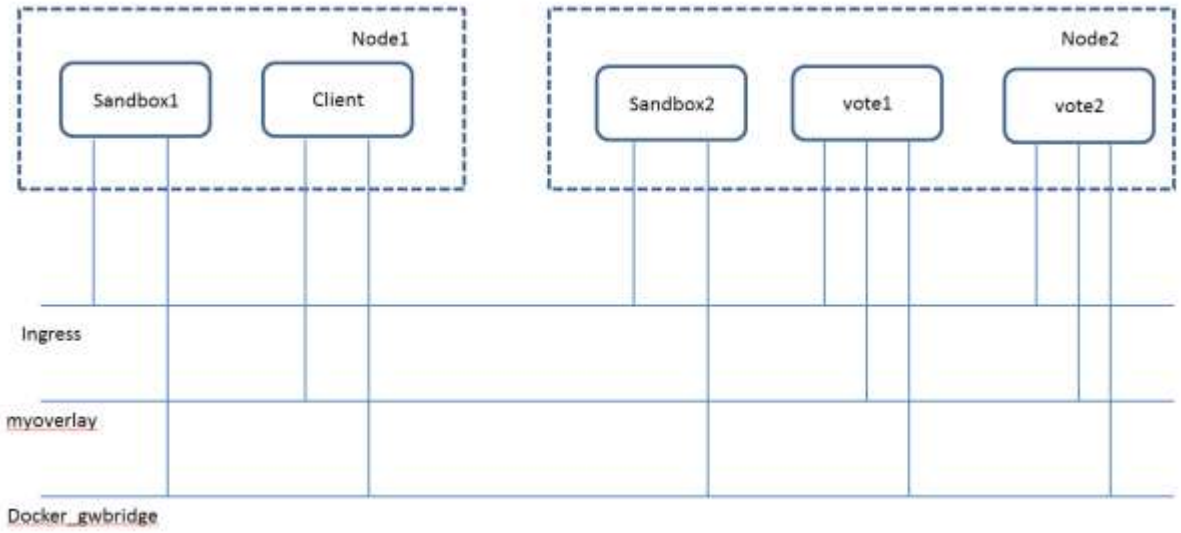


Accessing “vote” service using “ingress” network externally



Swarm Application - Networking detail

- Sandboxes and “vote” containers are part of “ingress” network and it helps in routing mesh.
- “client” and “vote” containers are part of “overlay1” network and it helps in service connectivity.
- All containers are part of the default “docker_gwbridge” network. This helps for external access when services gets exposed using publish mode “host”



Docker Network debug commands

- Basic Swarm debugging:

Docker node ls

- Service and Container debugging:

Docker service logs <service name/id>
Docker service inspect <service name/id>
Docker service ls
Docker service ps <service name/id>
Docker container logs <container name/id>
Docker container inspect <container name/id>

- Network debugging:

Docker network inspect <network name/id>

Use "-v" option for verbose output

Troubleshooting using debug container

- All Linux networking tools are packaged inside “nicolaka/netshoot”(https://github.com/nicolaka/netshoot) container. This can be used for debugging.
- Using this debug container avoids installation of any debug tools inside the container or host.
- Linux networking tools like tcpdump, netstat can be accessed from container namespace or host namespace.

Capture port 80 packets in the Container:

docker run -ti --net container:<containerid> nicolaka/netshoot

tcpdump –i eth0 –n port 80

Capture vxlan packets in the host:

docker run -ti --net host nicolaka/netshoot

tcpdump –i eth1 –n port 4789

- Debug container can also be used to get inside container namespace, network namespace and do debugging. Inside the namespace, we can run commands like “ifconfig”, “ip route”, “brctl show” to debug further.

Starting nsenter using debug container:

docker run -it --rm -v /var/run/docker/netns:/var/run/docker/netns --privileged=true nicolaka/netshoot

Getting inside container or network namespace:

nsenter –net /var/run/docker/netns/<networkid> sh

Useful commands inside Network namespace

Interface list:
Ifconfig

Bridge list:
brctl show

Route table:
Ip route show

Link details:
ip -d link show

Arp table:
ip -s neighbor show

Encapsulation detail:
bridge fdb show

Capture packets:
Tcpdump -I <interface> -n port <port number>

Iptables nat and mangle rule:
iptables -nvL -t nat
iptables -nvL -t mangle

Check load balancing:
ipvsadm

References

- Troubleshooting container
- Docker Networking Overview
- Docker Networking – common issues and troubleshooting techniques
- Docker blogs by me