KIRATECH®
WE DEVOPS IT

Overlay Network
Multi Docker Host Networking

Marco Bizzantino
CTO @ Kiratech
marco.bizzantino@kiratech.it

@bizzam
#containerday

docker

Understand Docker container networks

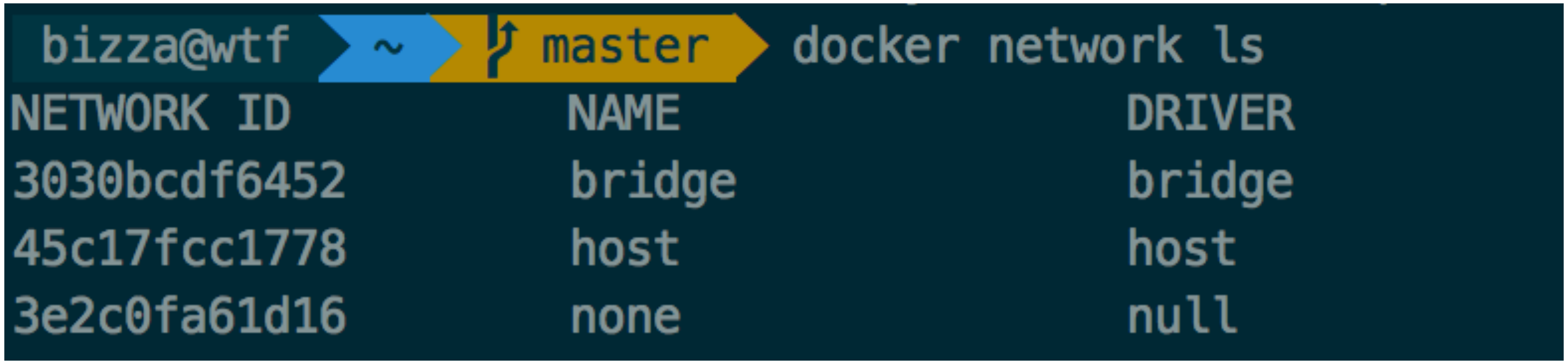
- Networks, by definition, provide complete isolation for containers
- It’s important to have control over the networks
- Docker container networks give you that control

Docker networking model

- Containers do not have a public IPv4 address
- They are allocated a private address
- Services running on a container must be exposed port by port
- Container ports have to be mapped to the host port to avoid conflicts

Default Network

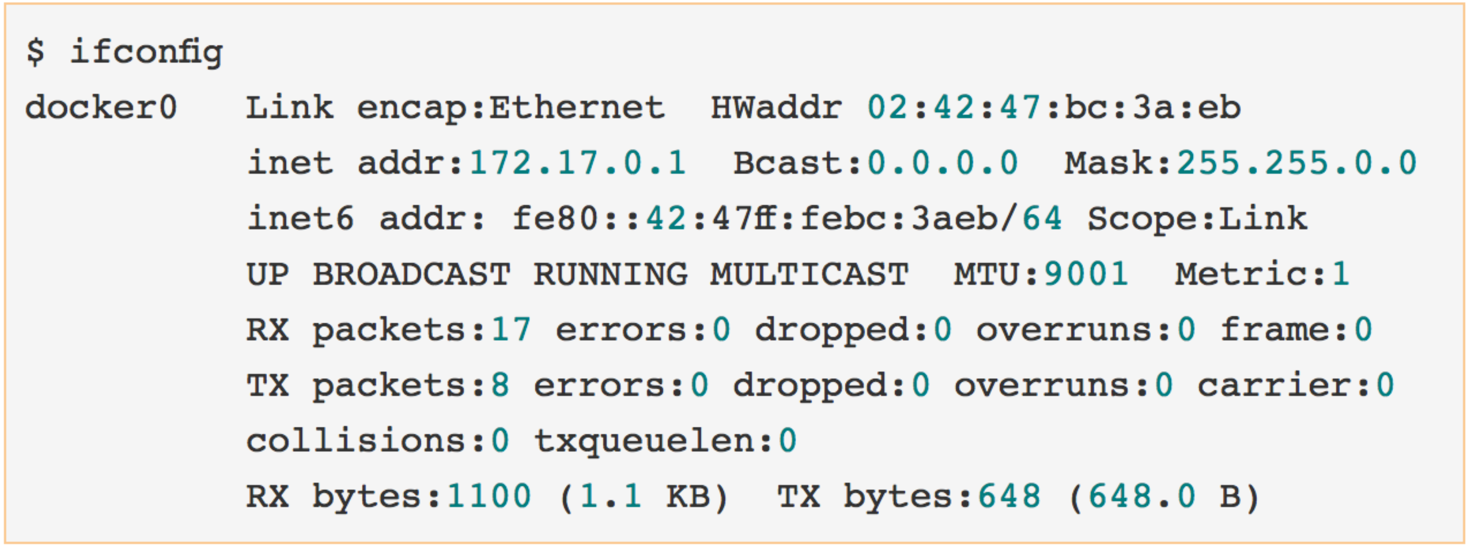
- Docker installation creates three networks automatically
- You can use --net flag to specify which network you want to run a container on



```
bizza@wtf  ~  master  docker network ls
NETWORK ID          NAME                DRIVER
3030bcdf6452        bridge              bridge
45c17fcc1778        host                host
3e2c0fa61d16        none                null
```

Bridge Network

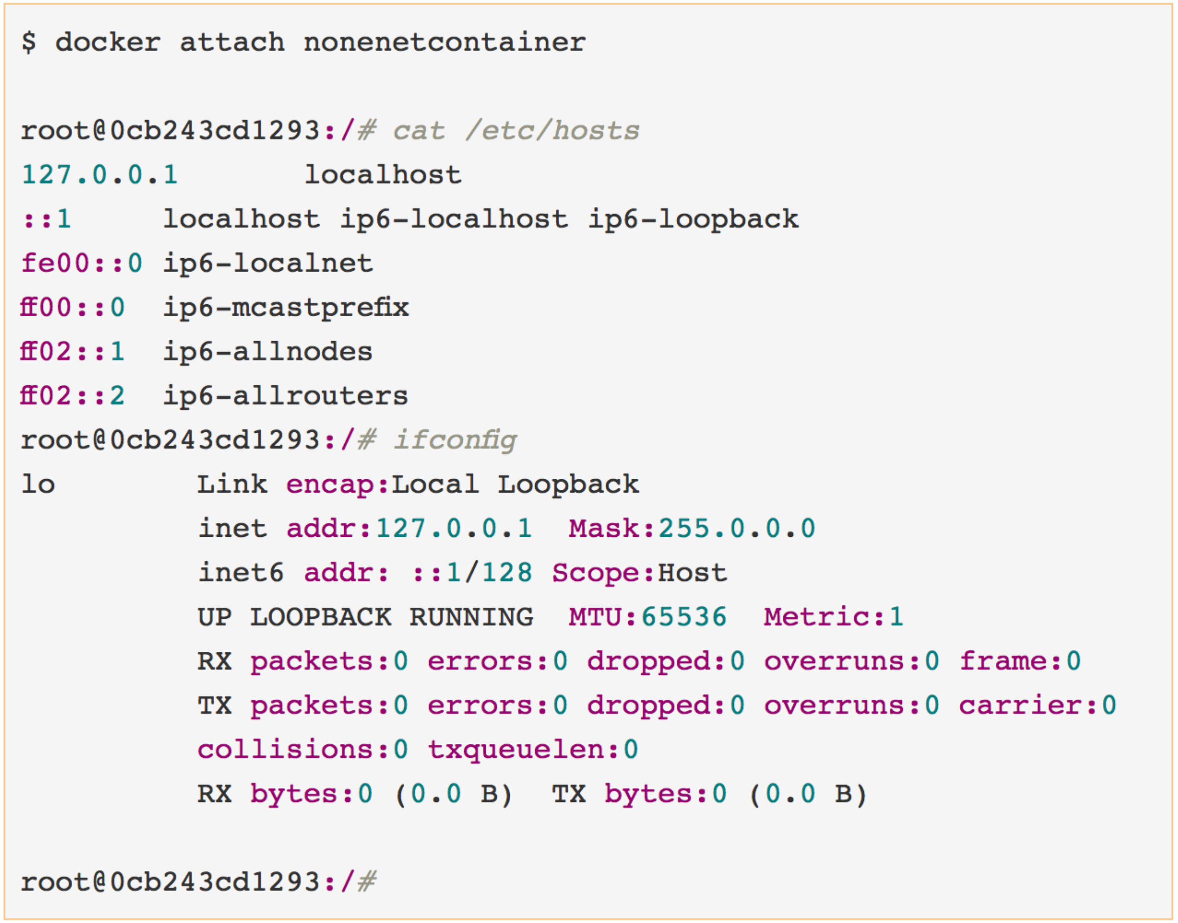
- Is the docker0 network present in all Docker installations
- All containers by default connects to it
- Part of host’s network stack
- docker0 is assigned a random IP address and subnet from the private range defined by RFC 1918



```
$ ifconfig
docker0   Link encap:Ethernet  HWaddr 02:42:47:bc:3a:eb
          inet addr:172.17.0.1  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::42:47ff:febc:3aeb/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:9001  Metric:1
          RX packets:17 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1100 (1.1 KB)  TX bytes:648 (648.0 B)
```

None Network

- Container-specific network stack
- Container attached lacks a network interface



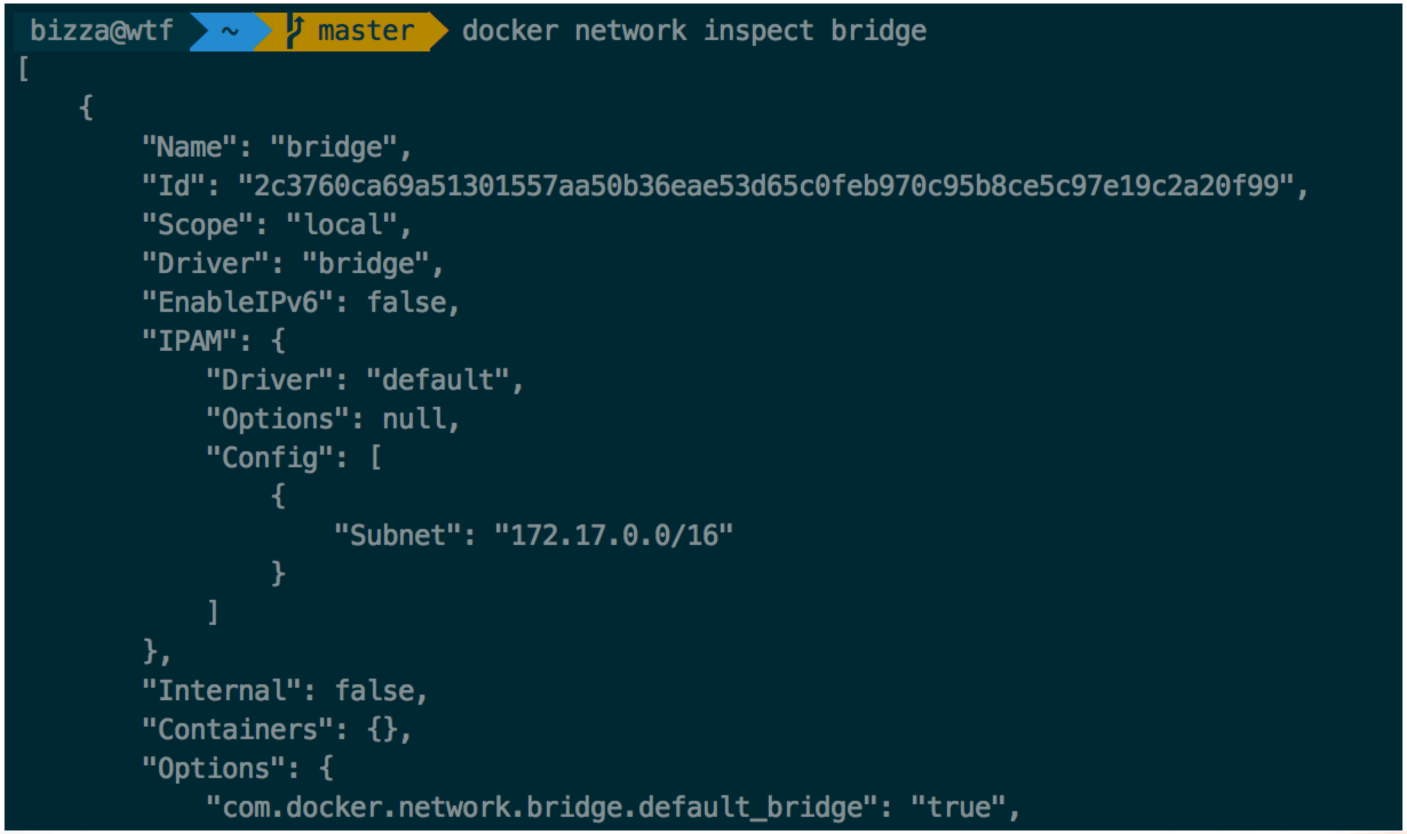
```
$ docker attach nonenetcontainer

root@0cb243cd1293:/# cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
root@0cb243cd1293:/# ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@0cb243cd1293:/#
```

Host Network

- Adds a container on the host network stack
- Network configuration inside the container is identical to the host

Check container networking properties

The docker network inspect command returns information about a network



```
bizza@wtf  ~  master  docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "2c3760ca69a51301557aa50b36eae53d65c0feb970c95b8ce5c97e19c2a20f99",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16"
                }
            ]
        },
        "Internal": false,
        "Containers": {},
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
```

Network summary

- Docker containers run in a subnet provisioned by the docker0 bridge on the host machine
- We can create our own bridge or different network to run containers on
- Auto mapping of container ports to host ports only applies to the port numbers defined in the Dockerfile EXPOSE instruction

Multi-host networking

- Containers running on different hosts cannot communicate with each other without mapping their TCP ports to the host’s TCP ports
- Multi-host networking allows these containers to communicate without requiring port mapping
- The Docker Engine supports multi host networking natively out of the box via the **overlay** network driver

Multi-host networking

Requirements for creating an overlay network

- Access to a key-value store
- A cluster of hosts connected to the key-value store
- All hosts must have Kernel version 3.16 or higher
- Docker Engine properly configured on each host

Overlay network

- overlay network driver supports multi-host networking natively out-of-the-box
- Based on libnetwork, a built-in VXLAN-based overlay network driver, and Docker’s libkv library
- The overlay network requires a valid key-value store service
- The Docker hosts must be able to communicate
    - udp port 4789 Data plane (VXLAN)
    - tcp/udp port 7946 Control plane

Key-value store

Stores information about the network state including

- Discovery
- Endpoints
- IP addresses

Supported options

- Consul
- Zookeeper (Distributed store)
- Etcd
- BoltDB (Local store)

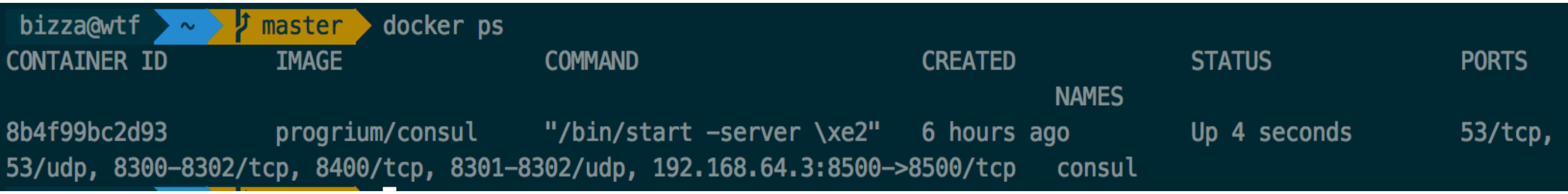
Setup key-value store

On your Master Node

Run consul in a container with the following command

```
docker run -d -p 8500:8500 -h consul --name consul \
     progrium/consul -server -bootstrap
```

Check that consul is running and that port 8500 is mapped to the host using docker ps



```
bizza@wtf  ~  master  docker ps
CONTAINER ID        IMAGE               COMMAND                       CREATED             STATUS              PORTS
                                                                                NAMES
8b4f99bc2d93        progrium/consul     "/bin/start -server \xe2"    6 hours ago         Up 4 seconds        53/tcp,
53/udp, 8300-8302/tcp, 8400/tcp, 8301-8302/udp, 192.168.64.3:8500->8500/tcp    consul
```

Configure Docker Engines

The Docker Engine on each node needs to be configured to:

- Listen on TCP port 2375
- Use the Consul key-value store on our master node

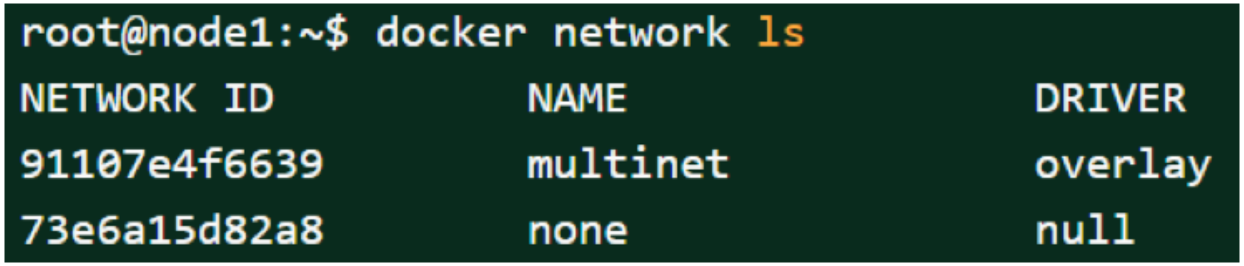
Modify the DOCKER_OPTS variable

```
DOCKER_OPTS="-H tcp://0.0.0.0:2375 \
        -H unix:///var/run/docker.sock \
        --cluster-store=consul://<Master Node IP>:8500/network \
        --cluster-advertise=eth0:2375"
```

Configure the Overlay network

Create an overlay network on one of the machines in the Swarm

docker network create -d overlay –subnet 10.10.2.0/24 multinet



```
root@node1:~$ docker network ls
NETWORK ID          NAME                DRIVER
91107e4f6639        multinet            overlay
73e6a15d82a8        none                null
```

Running containers on a multi-host network

To run a container on the multi-host network, you just need to specify the network name on the docker run command. For example:

```
docker run -itd --name c1 --net
multinet busybox
```

Can run containers from any host connected to the network

Container will be assigned an IP address from the subnet of your multi-host network

Running containers on a multi-host network

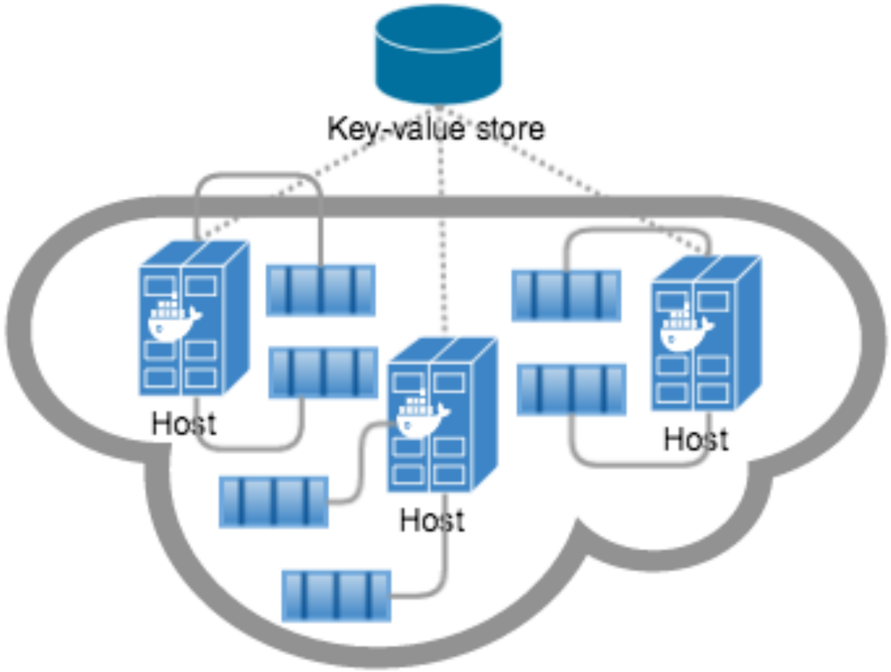
The first time an overlay network is created on any host, Docker also creates another network called **`docker_gwbridge`**

The docker_gwbridge network provides external access for containers

All TCP/UDP ports are open on an overlay network and thus, it is not necessary to map container ports to host ports in order for containers to communicate

Overlay Network

Once connected, each container has access to all the containers in the network regardless of which Docker host the container was launched on.



KIRATECH WE DEVOPS IT

Container discovery

- The docker daemon contains an embedded DNS server
- Containers must run with a name (using the `--name` option). This maps to the IP address on the network the container is connected to.
- When a container is added to a multi-host network, all other hosts will be able to discover it via the DNS server

Container discovery

- Container may have any number of aliases on a network
- Containers may have different aliases on different networks, set using the `--alias` option on `network connect`
- If the embedded DNS server is unable to resolve the request it will be forwarded to any external DNS servers configured for the container

Multi-host Network Summary

- An overlay (multi-host) network requires a key/value store
- Containers added to a multi-host network are discoverable by other containers, as long as the container name/alias has been specified
- Containers on different hosts can communicate with each other without exposing any ports if the hosts are part of the same overlay network

Macvlan and Ipvlan Network Drivers

- complete control of layer 2 VLAN tagging and even Ipvlan L3 routing for users interested in underlay network integration
- container attached directly to the Docker host interface
- easy access for external facing services as there is no port mappings
- still experimental

More informations:
https://github.com/docker/docker/blob/master/experimental/vlan-networks.md

Thank you

KIRATECH®

WE DEVOPS IT