**Understanding Docker Networking Drivers and their use cases**

Applications requirements and networking environments are diverse and sometimes opposing forces. In between applications and the network sits Docker networking, affectionately called the Container Network Model or CNM. It’s CNM that brokers connectivity for your Docker containers and also what abstracts away the diversity and complexity so common in networking. The result is portability and it comes from CNM’s powerful network drivers. These are pluggable interfaces for the Docker Engine, Swarm, and UCP that provide special capabilities like multi-host networking, network layer encryption, and service discovery.

Naturally, the next question is which network driver should I use? Each driver offers tradeoffs and has different advantages depending on the use case. There are built-in network drivers that come included with Docker Engine and there are also plug-in network drivers offered by networking vendors and the community. The most commonly used built-in network drivers are bridge, overlay and macvlan. Together they cover a very broad list of networking use cases and environments. For a more in depth comparison and discussion of even more network drivers, check out the Docker Network Reference Architecture.

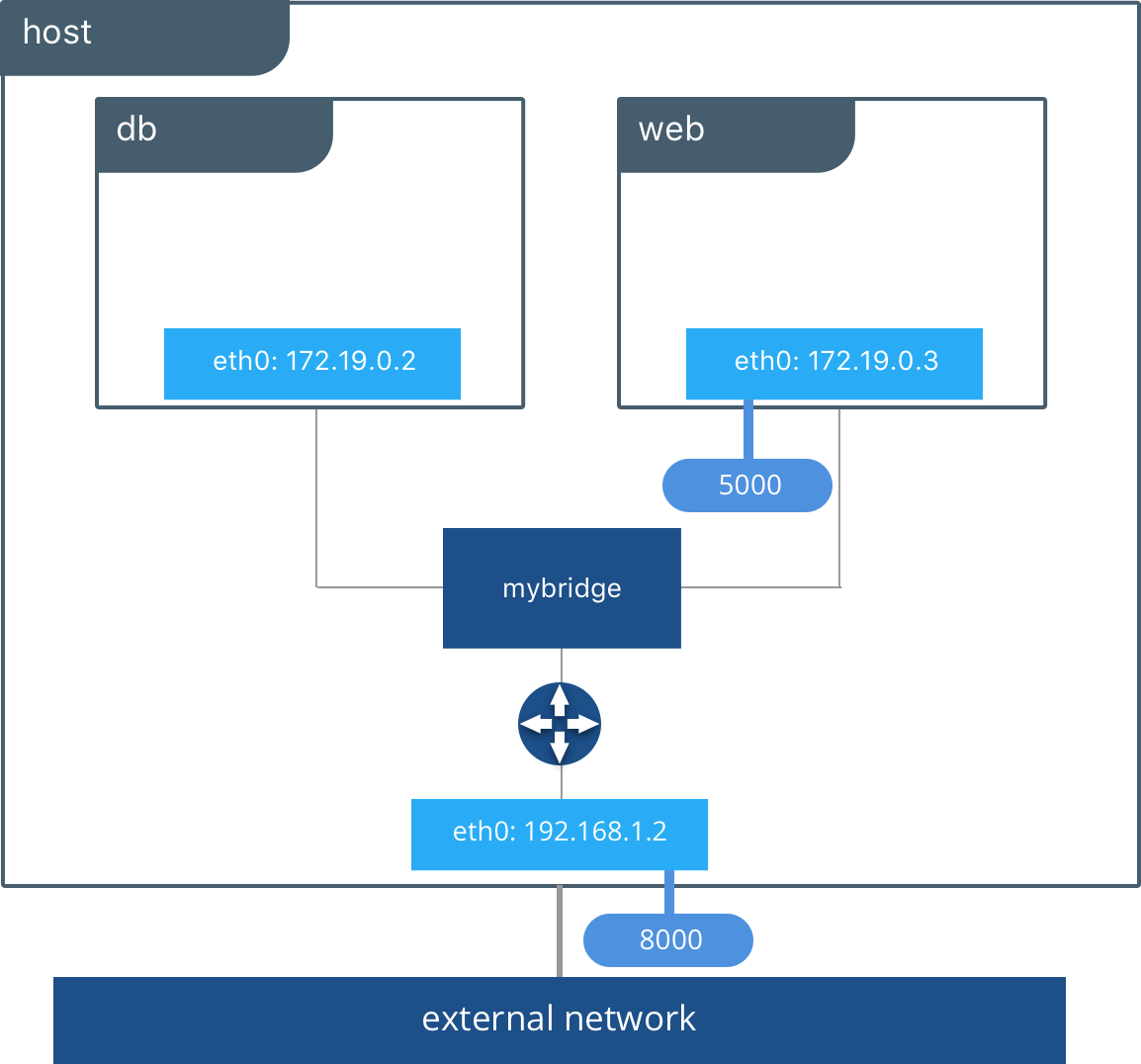
**Bridge Network Driver**

The bridge networking driver is the first driver on our list. It’s simple to understand, simple to use, and simple to troubleshoot, which makes it a good networking choice for developers and those new to Docker. The bridge driver creates a private network internal to the host so containers on this network can communicate. External access is granted by exposing ports to containers. Docker secures the network by managing rules that block connectivity between different Docker networks.

Behind the scenes, the Docker Engine creates the necessary Linux bridges, internal interfaces, iptables rules, and host routes to make this connectivity possible. In the example highlighted below, a Docker bridge network is created and two containers are attached to it. With no extra configuration the Docker Engine does the necessary wiring, provides service discovery for the containers, and configures security rules to prevent communication to other networks. A built-in IPAM driver provides the container interfaces with private IP addresses from the subnet of the bridge network.

In the following examples, we use a fictitious app called pets comprised of a web and db container. Feel free to try it out on your own UCP or Swarm cluster. Your app will be accessible on `<host-ip>:8000`.

***docker network create -d bridge mybridge docker run -d --net mybridge --name db redis docker run -d --net mybridge -e DB=db -p 8000:5000 --name web chrch/web***



Our application is now being served on our host at port 8000. The Docker bridge is allowing web to communicate with db by its container name. The bridge driver does the service discovery for us automatically because they are on the same network. All of the port mappings, security rules, and pipework between Linux bridges is handled for us by the networking driver as containers are scheduled and rescheduled across a cluster.

The bridge driver is a **local scope**driver, which means it only provides service discovery, IPAM, and connectivity on a single host. Multi-host service discovery requires an external solution that can map containers to their host location. This is exactly what makes the overlay driver so great.

**Overlay Network Driver**

The built-in Docker overlay network driver radically simplifies many of the complexities in multi-host networking. It is a **swarm scope**driver, which means that it operates across an entire Swarm or UCP cluster rather than individual hosts. With the overlay driver, multi-host networks are first-class citizens inside Docker without external provisioning or components. IPAM, service discovery, multi-host connectivity, encryption, and load balancing are built right in. For control, the overlay driver uses the encrypted Swarm control plane to manage large scale clusters at low convergence times.

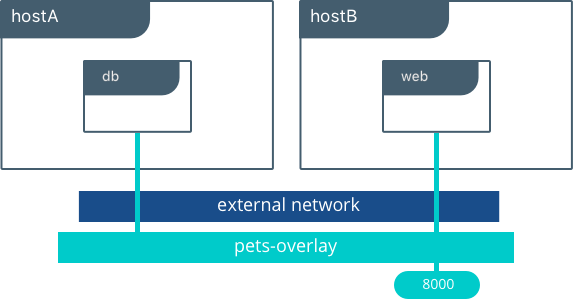
The overlay driver utilizes an industry-standard VXLAN data plane that decouples the container network from the underlying physical network (the *underlay*). This has the advantage of providing maximum portability across various cloud and on-premises networks. Network policy, visibility, and security is controlled centrally through the Docker Universal Control Plane (UCP).

In this example we create an overlay network in UCP so we can connect our web and db containers when they are living on different hosts. Native DNS-based service discovery for services & containers within an overlay network will ensure that web can resolve to db and vice-versa. We turned on encryption so that communication between our containers is secure by default.  Furthermore, visibility and use of the network in UCP is restricted by the permissions label we use.

UCP will schedule services across the cluster and UCP will dynamically program the overlay network to provide connectivity to the containers wherever they are. When services are backed by multiple containers, VIP-based load balancing will distribute traffic across all of the containers.

Feel free to run this example against your UCP cluster with the following CLI commands:

**docker network create -d overlay --opt encrypted pets-overlay docker service create --network pets-overlay --name db redis docker service create --network pets-overlay -p 8000:5000 -e DB=db --name web chrch/web**



In this example we are still serving our web app on port 8000 but now we have deployed our application across different hosts. If we wanted to scale our web containers, Swarm & UCP networking would load balance the traffic for us automatically.

The overlay driver is a feature-rich driver that handles much of the complexity and integration that organizations struggle with when crafting piecemeal solutions. It provides an out-of-the-box solution for many networking challenges and does so at scale.

**MACVLAN Driver**

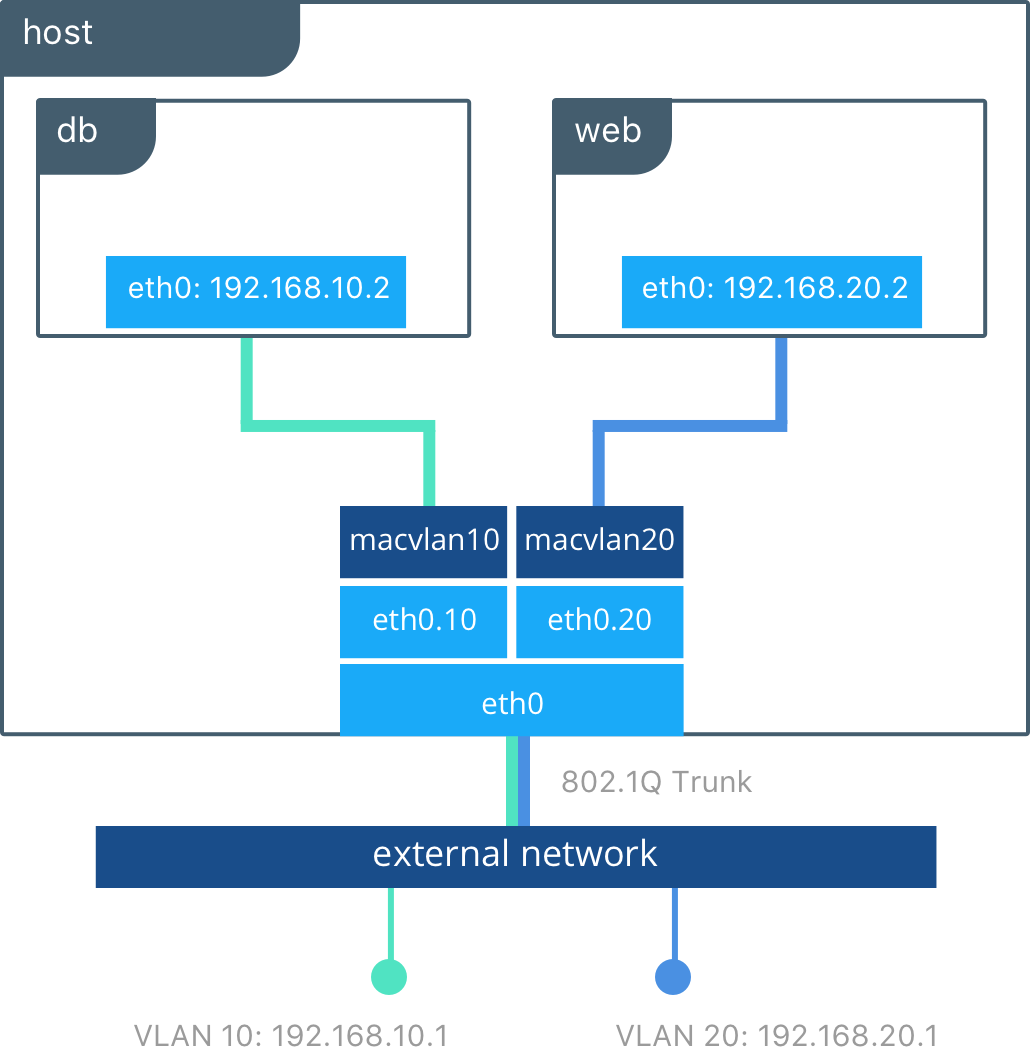
The macvlan driver is the newest built-in network driver and offers several unique characteristics. It’s a very lightweight driver, because rather than using any Linux bridging or port mapping, it connects container interfaces directly to host interfaces. Containers are addressed with routable IP addresses that are on the subnet of the external network.

As a result of routable IP addresses, containers communicate directly with resources that exist outside a Swarm cluster without the use of NAT and port mapping. This can aid in network visibility and troubleshooting. Additionally, the direct traffic path between containers and the host interface helps reduce latency. macvlan is a local scope network driver which is configured per-host. As a result, there are stricter dependencies between MACVLAN and external networks, which is both a constraint and an advantage that is different from overlay or bridge.

The macvlan driver uses the concept of a parent interface. This interface can be a host interface such as eth0, a sub-interface, or even a bonded host adaptor which bundles Ethernet interfaces into a single logical interface. A gateway address from the external network is required during MACVLAN network configuration, as a MACVLAN network is a L2 segment from the container to the network gateway. Like all Docker networks, MACVLAN networks are segmented from each other – providing access within a network, but not between

networks.

The macvlan driver can be configured in different ways to achieve different results. In the below example we create two MACVLAN networks joined to different subinterfaces. This type of configuration can be used to extend multiple L2 VLANs through the host interface directly to containers. The VLAN default gateway exists in the external network.



The db and web containers are connected to different MACVLAN networks in this example. Each container resides on its respective external network with an external IP provided from that network. Using this design an operator can control network policy outside of the host and segment containers at L2. The containers could have also been placed in the same VLAN by configuring them on the same MACVLAN network. This just shows the amount of flexibility offered by each network driver.

Portability and choice are important tenants in the Docker philosophy. The Docker Container Network Model provides an open interface for vendors and the community to build network drivers. The complementary evolution of Docker and SDN technologies is providing more options and capabilities every day.