Docker containers and Macvlan

5 minute read

A previous post describes how to use Ansible to create and manage docker services under systemd. After a few days it became clear that Ansible, systemd and Docker are working well together, but the network isn't scaling.

Docker networking

The docker networking strategy that is most often used is a type called 'bridged'. While named bridged, it could be more accuratly called PAT (Port Address Translation). In other words, to access the docker container you access the IP of the host the container a specific port that is mapped back tot he container. This worked well for one of two services, but in my experience scales poorly when running multiple different types of containers that often utilize the same ports. One then needs to remember that their Unifi controller is on port 9443 instead of the default 8443, etc. After about 4 containers this was becoming a struggle and I needed to find a better solution.

Macvlan

Recent versions Docker and linux kernels support another type of networking called Macvlan. Docker describes Macvlan (https://docs.docker.com/engine/userguide/networking/get-started-macvlan/)

> *Macvlan is a new twist on the tried and true network virtualization technique. The Linux implementations are extremely lightweight because rather than using the traditional Linux bridge for isolation, they are simply associated to a Linux Ethernet interface or sub-interface to enforce separation between networks and connectivity to the physical network.*
>
> *Macvlan offers a number of unique features and plenty of room for further innovations with the various modes. Two high level advantages of these approaches are, the positive performance implications of bypassing the Linux bridge and the simplicity of having less moving parts. Removing the bridge that traditionally resides in between the Docker host NIC and container interface leaves a very simple setup consisting of container interfaces, attached directly to the Docker host interface. This result is easy access for external facing services as there is no port mappings in these scenarios.*

This sounds great! Every container gets its own IP, no more tedious port mappings, no cheat sheet describing which service utilizes which port.

What is the catch?

The catch is that the host the containers are running on, by default, cannot talk to any of the containers. The architectural reasons for this are not really clear to me, the docker documentation says this:

> *That traffic is explicitly filtered by the kernel modules themselves to offer additional provider isolation and security.*

It certainly does provide isolation! An isolation that is not very useful for me. I need my host to talk to the containers for multple reasons. I have some services in containers and others are running on the host and they need to commmunicate.

How do you fix this failure to communicate?

This is where it got tricky. Nothing I could find worked for me. The docker Macvlan documentation says:

> *A macvlan subinterface can be added to the Docker host, to allow traffic between the Docker host and containers. The IP address needs to be set on this subinterface and removed from the parent address.*

But the example they provided did not work for me, I still had no connectivity between the containers and the host or if I could get connectivity between the containers and the hosts I could no longer contact the host from other hosts. Potentially part of my issue is that I was only carving out part of my 10.0.0.0/24 for docker containers. I eventually found this post (http://noyaudolive.net/2012/05/09/lxc-

and-macvlan-host-to-guest-connection/) documenting how to do macvlan with lxc. This also didn't work for me, but after reading it, I actually understood what had to happen. I had to remove the IP and all routing from my ethernet adapter and treat the phsyical host just like any container. Its nework configuration would be just like the container using the macvlan interface!

Macvlan all the things

The steps to enable communication between all devices on the subnet, the hosts, and the docker containers are:

1. Remove IP and routing from the physical adapter, but still activate the adapter.
2. Configure a macvlan adapter on the physical host, attached to the physical dapter with the physical adapters old IP and routing information.
3. Configure macvlan for docker
4. Change docker containers to use macvlan and assign each an IP.

Simplify the physical adapter

This was my original configuration for the adapter:

```
auto ens32
iface ens32 inet static
address 10.0.0.16
netmask 255.255.255.0
gateway 10.0.0.1
dns-nameservers 10.0.0.1
```

This was my configuraiton after removing all IPs and routing from the adapter:

```
auto ens32
iface ens32 inet manual
```

Note that the interface is changed from static to manual. This pemits enabling the interface without an IP.

Create the linked macvlan adapter

I used Ansible create and manage the new adapter.

```
# Construct macvlan interface
- name: Construct macvlan interface
  blockinfile:
    path: /etc/network/interfaces
    block: |
       auto macvlan0
       iface macvlan0 inet static
         address 10.0.0.16
         netmask 255.255.255.0
         gateway 10.0.0.1
         dns-nameservers 10.0.0.1
         pre-up ip link add link ens32 name macvlan0 type macvlan mode bridge
```

This adds the following segment to the interfaces file.

```
# BEGIN ANSIBLE MANAGED BLOCK
auto macvlan0
iface macvlan0 inet static
  address 10.0.0.16
  netmask 255.255.255.0
  gateway 10.0.0.1
  dns-nameservers 10.0.0.1
  pre-up ip link add link ens32 name macvlan0 type macvlan mode bridge
# END ANSIBLE MANAGED BLOCK
```

After this you can do an `if up macvlan0` or reboot and the host should have connectivity to the network like it always had, but the routing will look like this:

```
route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         firewall.aocboc 0.0.0.0         UG    0      0        0 macvlan0
10.0.0.0        *               255.255.255.0   U     0      0        0 macvlan0
```

Note how all traffic is now going through the macvlan interface.

Configure macvlan for docker

Following the instructions from Docker (https://docs.docker.com/engine/userguide/networking/get-started-macvlan/), Ansible can be used to create the macvlan docker network interface for the containes to use.

```
- name: Create macvlan network
  docker_network:
    name: macnet
    driver: macvlan
    driver_options:
      parent: ens32
    ipam_options:
      subnet: '10.0.0.0/24'
      gateway: '10.0.0.1'
      iprange: '10.0.0.112/28'
```

This creates a docker network that can have 16 IPs starting at 10.0.0.112. Any container on this network will get its own IP and have full access to all ports.

Change container to use the new network

With the new Macvlan network, a few changes are needed to configure the container.

```
- name: Create sagetv docker service
  include_role:
    name: mhutter.docker-systemd-service
  vars:
    service_name: sagetv
    image: stuckless/sagetv-server-java8:latest
    volumes:
      - "/mnt/recordings/:/var/media"
      - "/mnt/movies/:/var/mediaext"
      - "{{ containers_mount }}:/opt/sagetv"
    args: >
      --net macnet
      --hostname=sage
       "--ip={{ lookup('dig', 'sage.aocboc.') }}"
```

The network name is changed to macnet, which is the Macvlan network constructed above. Additionally, the IP is set by using the dig lookup to find the IP address for the container.

Summary

It took several nights of exploration and learning about Macvlan to figure this all out. But now each container gets their own IP, all containers and hosts can talk to each other and no more port mapping is needed!

Reach out to me on twitter (https://twitter.com/boc_tothefuture)