nETHer0

Vicente De Luca ---nethero--- what happens instead giving soccer balls to Brazilian kids, give them servers and routers at an ISP to play ?
| Passion for computers and BASIC coding started around age 7. At 11' my father and I was launching one of the firsts Internet Service Providers in Brazil, after lot of fun supporting a BBS. Since there I consider myself mostly a network/jack-of-all-trades dude | Further evolved to build and run enterprise cloud DCs, as well hacking mostly with Ruby or Golang. | Offline I am married with Paola ♡ -- still learning about this :) -- Have interests on reading about philosophy, macro political and economical matters. During the weekends I usually try flying RC planes, when the weather permits, or exploring new spots in Ireland, lovely country that well received us as part of my super happy career at Zendesk.

Connecting Docker containers to production network - IP per container

Since Docker recently showed up simplifying the way to containeirize applications, compared to manually handle LXC and Network namespaces, many developers are excited to use it for accelerate code deployments by shipping self sufficient containers from their laptop to the production datacenter linux workers .

Docker currently supports network setup in three ways basically:
- bridge
- NAT (port mapping)
- host mapping

Simple adding containers to a bridge interface and providing next free IP works for a single worker, not for a cluster of workers.

NO NAT !!! Adds lots of operational overhead, difficulty with logs, risk of exhaustions, last not least, it don’t scale.

The host-mapping (–net host) tells docker to bind containers sockets at the worker network namespace (meh..), so its mostly acceptable when not running multiple different applications at the same worker. In this mode you share the available ports to all the containers, so be aware about risk of port exhaustions which might drive you crazy tweaking sysctl trying to mitigate.

I don’t want to get paged in the midnight related to issues at my containers networking, so better I can do is providing one IP per container. Even refrigerators and toaster have their own IP, why not my containers right? :)

One way is by Routing (Layer3) having the worker as the default gateway for containers, running a dynamic routing protocol that advertises the containers IP addresses to the network routers. There is a cool project - Calico - that implements BGP protocol and integrates with Docker. It sounds pretty neat, but generally it is seen as rocket science – And it might really be. Mostly this kind of approaches can be complicated to support and overkill.

Guided by the “simple is beautiful” mantra, this post won’t propose you fancy overlay networks or rocket science solutions. My weapons are simple linux Bridge (Layer2) + well know DHCP Server to assign IP Addresses. This approach allow preserving the robustness and reliability already found on existing network models, as well don’t create new security zones. If it ain’t broke, don’t fix it.

Some benefits of the proposed solution:
- all containers can communicate with all other containers without NAT
- all nodes can communicate with all containers (and vice-versa) without NAT

Vicente De Luca ---ne
what happens instead
soccer balls to Brazilia
give them servers and
at an ISP to play ?
| Passion for compute
BASIC coding started
age 7. At 11' my fathe
was launching one of
firsts Internet Service
Providers in Brazil, aft
fun supporting a BBS.
there I consider mysel
a network/jack-of-all-tr
dude | Further evolved
build and run enterpris
DCs, as well hacking
with Ruby or Golang. |
I am married with Pao
still learning about this
Have interests on reac
about philosophy, mac
political and economic
matters. During the we
I usually try flying RC
when the weather per
exploring new spots in
Ireland, lovely country
well received us as pa
super happy career at
Zendesk.

CONTACT
MY FRIEND @SYSH
ASK ME ANYTHI
RSS
ARCHIVE

- the IP that a container sees itself as is the same IP that others see it as, making service discovery easier
- able to run at same worker multiple containers exposing the same service port
- It supports IPv4 (DHCP), IPv6 (SLAAC or DHCPv6) as well dual stack
- It might simplify a future migration to a containers cluster manager like Kubernetes

Architecture

The proposed layout contemplates the linux workers running ethernet bridges, extending the network broadcast domain to all the containers. It supports assignment of one (or more) IPv4 and IPv6 addresses per container. For IPv4 it requires a DHCP Server, for IPv6 native SLAAC Router Advertisements.

I am relying on ISC BIND to dynamically assign leases to containers, and dynamically registering DNS A records i.e: container_id.app.mydomain.com

This architecture might be easily portable to service providers that supports multiple IPs via DHCP server (or IPv6 SLAAC), like AWS and others.

image

How it works?

So far Docker still don’t feature “advanced networking” to help us. Running a DHCP client from the container namespace might be the fastest approach, but facing the container as a “RPM package on steroids” this track breaks the idea of having the container as a self sufficient app and nothing else. As well containers don’t have init.

We’ll be considering the DHCP client running at the worker userspace, attached to the container virtual network interface. A bash script docker wrapper (network wrapper) needs to be executed after the container start to acquire the network information from the DHCP server and configure the container networking, also manage the lease renews and all DHCP RFC expectations.

network-wrapper is my simplified version +some hacks based on a super cool tool written by Jérôme Petazzoni to handle softwared defined networking for containers github.com/jpetazzo/pipework

Basically I have ripped off from pipework everything not needed and added simple methods to generate unique locally administered MACs, and an inspect feature to spit container network information in JSON format, exactly as docker inspect does when we use it for handling the network setup.

Execution algorithm:

- Generate a locally unique MAC address (RFC compliance)
  MAC scheme: XX:IP:IP:IP:YY:ZZ where,
  XX = restricted random to always match unicast locally administered octets – two least significant bits = 10
  IP:IP:IP = worker IPv4 three last octets in HEX
  YY:ZZ = random portion

- Creates a new linux network namespace for the container
- Associate the container veth pair with the worker bridge interface (br0) and a new container eth0 having the generated unique mac
- At the worker runtime, execute and maintain running the DHCP client (dhclient or udhcpc) for the created network namespace

How to use the network wrapper for spining up a new container:

```
root@docker-lab vdeluca]# network-wrapper
Syntax:
network-wrapper inspect[-all] <guestID>
network-wrapper <HostOS-interface> [-i Container-interface] <guestID> dhcp [macaddr]

[root@docker-lab vdeluca]# network-wrapper br0 $(docker run --net none -d nginx) dhcp
{
    "Container": "4b94d0da2047",
        "NetworkSettings": {
          "HWAddr": "c6:64:50:63:3b:86",
          "Bridge": "br0",
          "IPAddress": "10.100.80.190",
          "IPPrefixLen": "24",
          "Gateway": "10.100.80.1",
          "DNS": "10.100.64.9, 10.100.66.9",
          "DHCP_PID": "23384"
} }
[root@docker-lab vdeluca]# curl 4b94d0da2047.intranet.mydomain.com
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
```

gistfile1.txt hosted with ♡ by GitHub view raw

Note: docker run –net none.
This step tells docker to not handle the network setup. Otherwise the container will fail on network due to having multiple default routes – one from docker, other from dhcp
Make sure the applications are binding to 0.0.0.0, or [::] in case of IPv6

dhcp-garbagecollection

A watchdog tool for keep consistency between docker running containers and DHCP client running processes. Watch and compare docker ps PIDs with DHCP clients PIDs.

The existence of containers without its respective DHCP client process, or the existence of DHCP client process without a container will trigger an action to kill the zombie process/container.

network-wrapper files

https://github.com/zenvdeluca/network-wrapper

Requirements and Notes

Tested on Ubuntu 12.04 (udhcpc) and CentOS 7 (dhclient)
Packages: bridge-utils (required), syslinux (required), arping (recommended)

Notes for CentOS7:
The /sbin/dhclient-script comes with tweaks that avoid the DHCP client to correct configure the default route in my environment.
Replacing the dhclient-script with the CentOS 6 makes it perfectly works.