5.1 LXC Advanced Networking - Exposing Containers to the Network

Introduction

By default, your containers are accessible only from the host. For serious use you will want to expose some containers to the outside world. There are various ways of doing this. Currently I have settled on the following,

Port Forwarding using Static IPs with IP Tables - allows you to leverage your host's IP address (assuming it is public).

macvlan with Additional IP - allows you to have, a dedicated network interfaces (to the outside world) but actually only use one real physical network card. Unlike using a bridge this will not have the cpu overhead and need for your network card to work in promiscuous mode. This article builds on the work done in the introductory LXC article.

I actually use multiple techniques together.

> Make sure to change the password or better remove the default ubuntu account generated by the lxc creation script before making the container accessible to the Internet.

This will get better over time, ie compared to Solaris where you just tell the container to use a public IP address and the macvlan or bridging is done for you.

Port Forwarding using IP Tables

You might want to use one IP Address on the host and then map specific ports out from the containers. As a pre-requisite you will need to setup Static LXC Assigned IP address.

There are a number of ways to do this but I favour iptables.

In this example we setup Apache which runs on port 80 in the container which has been assigned the static IP 10.0.3.10.

Make sure the host is not listening on port 80,

```
netstat -an | grep LISTEN | grep 80
tcp6       0      0 fe80::2cd7:eff:fea3::53 :::*                    LISTEN
```

Next determine your hosts network card name associated with the hosts IP address,

```
ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:0d:3a:02:e6:8a
          inet addr:10.0.0.4  Bcast:10.0.0.255  Mask:255.255.255.0
          inet6 addr: fe80::20d:3aff:fe02:e68a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:964479 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1199824 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:333377198 (333.3 MB)  TX bytes:1757835361 (1.7 GB)


# ... more get's displayed
```

In this case, I know my host's public IP6 address (put reference of how to convert to ip4 or reverse lookup on dns IP6 address) is fe80::20d:3aff:fe02:e68a/64 and see it as the first entry. This let's us know the network card name is **eth0**. In Ubuntu it will normally be **eth0** or **en s33**.

While on the host issue these commands, (TBD, look at making own named chain to distinguish the rules)

```
# Immediately enable port forwarding rule, but this is not persistent on
reboot.
# This is for all external connections coming in.
sudo iptables -t nat -A PREROUTING -p tcp -i eth0 --dport 80 -j DNAT
--to-destination 10.0.3.10:80

# If any containers initiate outbound requests, they will appear to
comewithin the hosts own IP and will not work.
# Allow container internal initiated requests to be converted using 'source
NAT',
#
http://blog.codeaholics.org/2013/giving-dockerlxc-containers-a-routable-ip
-address/ figuring out
```

Now traffic on port 80 on the host will be forwarded to port 80 in the container IP specified. You can see your rules, (note I got to review output again on a clean machine).

```
sudo iptables -t nat -L -n -v
Chain PREROUTING (policy ACCEPT 15 packets, 957 bytes)
 pkts bytes target     prot opt in     out     source
destination
    1    64 DNAT       tcp  --  eth0   *       0.0.0.0/0
0.0.0.0/0           tcp dpt:80 to:10.0.3.10:80

Chain INPUT (policy ACCEPT 1 packets, 229 bytes)
 pkts bytes target     prot opt in     out     source
destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source
destination

Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source
destination
 695K   53M MASQUERADE  all  --  *      *       10.0.3.0/24
!10.0.3.0/24
```

You might want to delete the rule. Again, note this is not persistent upon reboot. Execute the exact same command to add, but change -A to -D,

```
# Immediately delete rule created, but this is not persistent on reboot.
sudo iptables -t nat -D PREROUTING -p tcp -i eth0 --dport 80 -j DNAT
--to-destination 10.0.3.10:80
```

If you test the rule and like it. Make it persistent,

```
# To make persistent.
sudo apt-get install iptables-persistent # Tool replaces lots of manual
work documented here, https://help.ubuntu.com/community/IptablesHowTo
sudo /etc/init.d/iptables-persistent save
sudo /etc/init.d/iptables-persistent reload
```

Going further you can actually remap ports too. In this example, I map port 23 on the host server to the 22 on the container,

```
sudo iptables -t nat -D PREROUTING -p tcp -i eth0 --dport 22 -j DNAT
--to-destination 10.0.3.11:23
```

bridge with Additional IP Address

...

How to adjust LXD,

https://insights.ubuntu.com/2015/11/10/converting-eth0-to-br0-and-getting-all-your-lxc-or-lxd-onto-your-lan/

Also rever to my own article on bridge networking.

macvlan with Additional IP Address

Here's the Use Case,

> For further isolation you may have purchased an additional IP address. For example run Apache on your host using port 80 and then also run Apache inside one of your containers also requiring port 80. Since it is not possible to have port 80 twice on the same IP you opt to purchase a second IP.
>
> In most hosting services this is not expensive, but they will likely not give you a dedicated network card. In this example we purchase an additional static IP from the hosting company and use the same network card as the host.

The most viable options, I understand are using bridge or a dedicated vlan.

With macvlan you configure the container to directly use the public IP address without the overhead of changing the network card to promiscuous mode and have a lower CPU overhead. Once setup the macvlan gets it's own MAC address. This only works if there are no restrictions on the network which set's static IPs based on the hosts' MAC address. Usually this is only the case with the initial primary IP provided by the hosting company.

The containers can reach the network and each other, but not the host. Even though the host may be on the same network. This is by design of macvlan. The container network cards (Sub-interfaces) on the same host (parent) cannot communicate with each other. ALL frames from sub-interfaces are forward out through the parent interface. Even physcial switches that reflect the frames back in will get dropped.

If you had previously assigned a static IP to the container using **/etc/lxc/dnsmasq.conf** make sure to remove the entry (I believe you also need to restart the host).

> One interesting **limitation** is that the containers cannot resolve by DNS to the original Public IP directly used by the host. I don't have a use case requiring this, so will resolve iif/when needed.

You must have,

- 1 macvlan mapped to 1 container interface
- each interface must have different static IP addresses

macvlan mac address

The first thing to do is to create a mac address for the macvlan interface we will create on the host.

According to this article the Second bit of the most significant Byte indicates if the MAC address is **locally** or **universally** administered. (I think locally is better for public facing so you do not collide with other mac addresses). When generating MAC addresses for macvlan, this bit should be set to '1', indicating it is locally administered. As such, translated to a mac address any of these numbers are locally managed and you will not need to worry about colliding with the larger Internet,

```
x2-xx-xx-xx-xx-xx
x6-xx-xx-xx-xx-xx
xA-xx-xx-xx-xx-xx
xE-xx-xx-xx-xx-xx
```

Right now I have not read enough to describe the math, but for now, use the hellion website to generate a Random Locally Administered Unicast MAC Address. In this example, I selected 8a:38:2a:cc:d7:aa because the last aa is easy to search on.

> You can not use the same MAC interface on multiple containers on the same host. Otherwise, you will not be able to start you container and receive the error message about your interface already being in use.

Command Line macvlan

You can use the command line inside your container to quickly create a macvlan and test, but it will disappear after reboot,

```
ip link add mvlan0 link eth0 address 8a:38:2a:cc:d7:aa type macvlan mode
bridge
ifconfig mvlan0 up
```

Create a Permanent macvlan on the Host

Add to the bottom of the **/etc/network/interfaces** file of the host,

```
# Creates a macvlan interface called macvlan0 without an IP address
iface mvlan0 inet manual
   pre-up ip link add mvlan0 link eth0 address 8a:38:2a:cc:d7:aa type
macvlan mode bridge
   post-down ip link del macvlan0
auto mvlan0
```

Notice that the MAC address is locally generated. The mavclan interface is actually not directly used and the MAC address will not actually register with anything. I am uncertain if it matters, so I have put in a static rather than generated MAC address out of preference (I don't like the idea of it changing on every boot). Not using a MAC address at all here might work too. If you the reader has time, let me know.

This macvlan0 is a placeholder on the host that will be used by a container interface. I have only purchased one additional static IP address so not all scenarios are tested.

If somebody gets to step 2 in below chart, please fill out for me. Otherwise, I'll try myself using my home network which is not exactly the same thing.

#	Host	container1	container2	Results
1	mvlan0	connect to mvlan0 with static-IP-1 with container2 off	connect to mvlan0 with static-IP-1 with container1 off	success
2	mvlan0	connect to mvlan0 with static-IP-1 with container2 on	connect to mvlan0 with static-IP-2 with container1 on	Not sure... I need to buy another static IP to test

Scenario 1 basically shows that a macvlan in the host may be used by multiple containers as long as only one host is on at a time.

Scenario 2 may or may not in which case we would want to create a scenario 3 by adding an additional macvlan mvlan1.

Look at your existing network cards,

```
ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:27:13:69:9c:9d
          inet addr:192.168.0.102  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::227:13ff:fe69:9c9d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1185139 errors:0 dropped:0 overruns:0 frame:0
          TX packets:779265 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:267212921 (267.2 MB)  TX bytes:164973471 (164.9 MB)
          Interrupt:20 Memory:fc400000-fc420000
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:125735 errors:0 dropped:0 overruns:0 frame:0
          TX packets:125735 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:10929290 (10.9 MB)  TX bytes:10929290 (10.9 MB)
lxcbr0    Link encap:Ethernet  HWaddr 00:00:00:00:00:00
          inet addr:10.0.3.1  Bcast:10.0.3.255  Mask:255.255.255.0
          inet6 addr: fe80::4483:50ff:feb5:bbe4/64 Scope:Link
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:1174 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1493 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:106747 (106.7 KB)  TX bytes:2243760 (2.2 MB)
```

Reboot your system to have the change take effect. Review your network card list,

```
ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:27:13:69:9c:9d
          inet addr:192.168.0.102  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::227:13ff:fe69:9c9d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:222 errors:0 dropped:0 overruns:0 frame:0
          TX packets:177 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:23794 (23.7 KB)  TX bytes:19751 (19.7 KB)
          Interrupt:20 Memory:fc400000-fc420000
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:534 errors:0 dropped:0 overruns:0 frame:0
          TX packets:534 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:42672 (42.6 KB)  TX bytes:42672 (42.6 KB)
lxcbr0    Link encap:Ethernet  HWaddr e2:2a:05:fb:9a:ab
          inet addr:10.0.3.1  Bcast:10.0.3.255  Mask:255.255.255.0
          inet6 addr: fe80::e02a:5ff:fefb:9aab/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 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:0 (0.0 B)  TX bytes:648 (648.0 B)
mvlan0    Link encap:Ethernet  HWaddr 8a:38:2a:cc:d7:aa
          inet6 addr: fe80::8838:2aff:fecc:d7aa/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:46 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:3607 (3.6 KB)  TX bytes:648 (648.0 B)
wlan0     Link encap:Ethernet  HWaddr 00:26:c6:81:f7:4a
          BROADCAST MULTICAST  MTU:1500  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:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Notice mvlan0 is present with the hardware address you specified.

Connect Container to macvlan on Host

Now one container may connect to the **mvlan0** interface on the host and they will get their IPs directly from the same network connected to the host (if DHCP) or you can assign static IPs inside the container that are reserved for you.

Make the container aware of the mvlan0 by modifying the config file located in **/var/lib/lxc/[container]/config**. In this example, I named the container web so the location will be **/var/lib/lxc/web/config**.

```
# The directory itself is root only so for ease of browsing you might want
to switch to root
sudo su -
cd /var/lib/lxc/[container]/config # change [container] to your container
name, in my case "web"
```

We will be adding a network card for the container. First step is to use the hellion website to generate a Random Locally Administered Unicast MAC Address. Then, modify the **config** file by adding a network card for the container.

```
# Template used to create this container:
/usr/share/lxc/templates/lxc-ubuntu
# Parameters passed to the template:

# For additional config options, please look at lxc.container.conf(5)
# Common configuration
lxc.include = /usr/share/lxc/config/ubuntu.common.conf

# Container specific configuration
lxc.rootfs = /var/lib/lxc/web/rootfs
lxc.mount = /var/lib/lxc/web/fstab
lxc.utsname = web
lxc.arch = amd64

# Network configuration

# macvlan for external IP
lxc.network.type = macvlan
lxc.network.macvlan.mode = bridge
lxc.network.flags = up
lxc.network.link = mvlan0
lxc.network.hwaddr = 00:16:3e:8d:4f:51
lxc.network.name = eth0

# Interface using LXC dhcp to communicate with other containers
lxc.network.type = veth
lxc.network.flags = up
lxc.network.link = lxcbr0
lxc.network.hwaddr = 00:16:3e:a2:7d:54
lxc.network.name = eth1
```

This tells the container to use the macvlan network interface mvlan0 which we created in the host and in turn map it to the container's eth0 interface.

I generated a new MAC address here by creating a temporary container, copying the mac address and then destroying the container. The LXC mac address looks like a universally administered number which worked with my hosting provider. For some reason the locally administered did not appear to work (I'll need to do one more test again when time permits to confirm).

There is a command line way of generating an LXC mac address this (actually documented somewhere in this LXC documentation) but I have not had a chance to try it.

Also, note that I kept the automatically generated interface eth0 and renamed it eth1. This will allow the container to communicate to other containers who are using the LXC DHCP provided IP address in the internal LXC network.

> There is probably a way to setup the static ip address here in this config file for the container. I am pretty sure I saw an option.

However, in the vein of individual container control, I would rather set that at the container level using steps outlined below.

Next, modify the **interfaces** file within the container to your selected static IP address. I find the fastest is from within the Host OS using root, **/var/lib/lxc/<container name>/rootfs/etc/network/interfaces** which in this case would be, **/var/lib/lxc/web/rootfs/etc/network/interfaces**,

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
address 64.73.220.117
gateway 64.73.220.1
netmask 255.255.255.0
dns-nameservers 216.15.129.205 216.15.129.206

auto eth1
iface eth1 inet dhcp
```

Above we now have two interfaces. The static interface is the public IP address purchased called eth0 and second is the internal LXC assigned address. The other entries can be obtained from Add Additional IP Addresses to Ubuntu Server.

Update dnsmasq

Make sure to check your dnsmasq and make modifications accordingly. Using the example since we modified a container that was already using dnsmasq we needed to change the original **/etc/lxc/dnsmasq.conf**,

```
# Specify static IPs per container name or mac address.
# Addresses should be outside of range in /etc/default/lxc-net (10.0.3.100
to 10.0.3.254)

dhcp-host=web,10.0.3.10
dhcp-host=app,10.0.3.20
dhcp-host=database,10.0.3.30
```

The new file looks like this,

```
# Specify static IPs per container name or mac address.
# Addresses should be outside of range in /etc/default/lxc-net (10.0.3.100
to 10.0.3.254)

# Container t01web uses macvlan for both a public and internal ip address
requiring reference by MAC Address.
dhcp-host=00:16:3e:a2:7d:54,10.0.3.10
dhcp-host=app,10.0.3.20
dhcp-host=database,10.0.3.30
```

Because the container called "web" now has two interfaces. 1 static, 1 we still want to assign through dnsmasq. In order for dnsmasq to know which interface to use you, must now specify by using the mac address contained in the **/var/lib/lxc/[container]/config**.

You need to shutdown containers and restart the host or flush the DNS Masq,

When editing **/etc/lxc/dnsmasq.conf**, changes do not take effect right away.

So **first** shutdown all your containers (Aug 2016, there is an issue that lxc-net does not restart unless all containers are shut off). Then restart lxc-net,

```
# Tried this and it did noot work
http://seminar.io/2014/07/27/dns-resolution-for-lxc-in-ubuntu-trusty/
sudo service lxc-net stop
sudo service lxc-net start
```

The other option is to shutdown all containers and restart the host operating system.

Multiple Interfaces

In most cases you will want multiple interfaces. In this example, we built a front-end container called "web", gave it a public IP address using mavlan. In addition, we create an "app" container which has an lxc internal IP address. In order for web and app to communicate, web must have a second interface that also uses an lxc provided internal IP address.

(...draw a diagram here...)

I got this all working, but need to document.

macvtap

This looks promising... The most prominent user of macvtap interfaces seems to be libvirt/KVM, which allows guests to be connected to macvtap interfaces. Doing so allows for (almost) bridged-like behaviour of guests but without the need to have a real bridge on the host, as a regular ethernet interface can be used as the macvtap's lower device.

References

Networking - https://help.ubuntu.com/lts/serverguide/lxc.html#lxc-network

Networking LXD More Detail - https://www.flockport.com/lxc-networking-guide/

INotes issue about bridge mode promiscuous mode - http://wiki.alpinelinux.org/wiki/LXC#Creating_a_LXC_container_without_modifying_your_network_interfaces

Getting macvlan working - https://www.flockport.com/lxc-macvlan-networking/

Learning macvlan and good background info - http://backreference.org/2014/03/20/some-notes-on-macvlanmacvtap/

Oracle article on details of container creation - http://docs.oracle.com/cd/E37670_01/E37355/html/ol_config_os_containers.html#ol_setup_fs_containers

Looks to be most comprehensive yet of what I want (multiple network cards with macvlan) - http://containerops.org/2013/11/19/lxc-networking/

How I figured out to create a macvlan - http://cyberiantiger.livejournal.com/24104.html

Not sure I can use comments here... need to investigate if it causes issues.

Generate mac address same way lxc does - http://giantdorks.org/alain/how-to-generate-a-unique-mac-address/

More in depth and discusses outbound NAT so containers can communicate to other container public IPs -http://blog.codeaholics.org/2013/giving-dockerlxc-containers-a-routable-ip-address/

Bridge versus Macvlan - http://hicu.be/bridge-vs-macvlan