CIAB LXD Internetworking
for Multi-Tenant, Multi-Node, Hybrid/Multi-Cloud

by Brian Mullan (bmullan.mail@gmail.com)

PREFACE

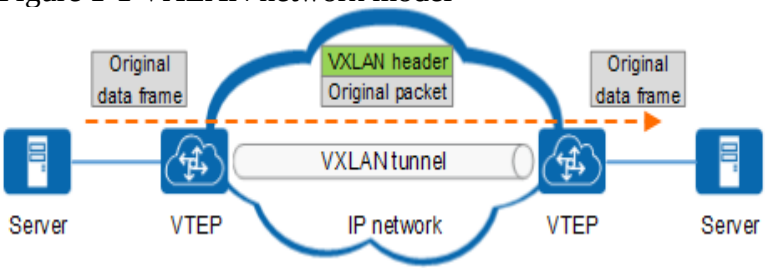
CIAB Multi-Tenant, Multi-Node, Multi-Cloud Internetworking

The CIAB internetworking architecture is implemented using several open source linux networking tools/apps, Routing and protocol capabilities including:

Free Range Routing (FRR) - FRR is an IP routing protocol suite for Linux and Unix platforms which includes protocol daemons for BGP, IS-IS, LDP, OSPF, PIM, and RIP.

VxLAN – **an overview of VxLAN and Linux**

Figure 1-1 VXLAN network model



A VXLAN network is a virtual Layer 2 network constructed on a Layer 3 network to enable communication of hosts at Layer 2.

BGP w/Route Reflectors – **a general overview**

BGP Route reflectors (RR) are one method to get rid of the full-mesh of IBGP peers in your network.

WireGuard – **WireGuard VPN** is now built into the Linux Kernel => 4.4

LXD – **LXD is a System Container Hypervisor**

VxWireGuard-Generator - **Utility to generate VXLAN over Wireguard mesh SD-WAN configuration**

Option

1. Implement and utilize a full-mesh, auto-learning VPN
2. Implement and utilize VXLAN and OVS bridges

This document is focused on Option #1.

Option #2 will be described in a separate document.

Goals of Project

1. An ideal solution to satisfy the goals of this project will include these success KPI factors:
2. Security and Secure communications
3. Open Source
4. Use LXD Containers and LXD Container technologies
5. Easy/simple installation, configuration and expansion
6. Multi-tenant capable (re in LXD Containers)
7. Multi-node (re Multi LXD Host/Server) capable
8. Multi-Cloud and Hybrid Cloud capable

Full Mesh, Auto-Learning VPN using WireGuard.

For full-mesh VPN I really liked features (TAP/TUN/Hub) incorporated in VpnCloud as well as its overall ease of configuration and use.

However, what won me over to WireGuard was that it was slightly better performing that VpnCloud although more involved in configuration & configuration updates when a new node joins a VPN.

The biggest advantage WireGuard had over VpnCloud was that WireGuard runs "in Kernel space".

Given that one of the main "goals" of the project is to enable LXD container this was the deciding factor.

Why?

LXD Containers on a Host all use the Host/Server Kernel. Since WireGuard runs "in Kernel space" that means that WireGuard itself only needs to be installed and run in the Host/Server for all LXD Containers on that Host/Server to utilize it.

Each LXD container would have to have the WireGuard utility programs installed in them and a configuration file for that Container's VPN but that is a minor task and takes a minimal amount of disk space overall.

WireGuard in the Host/Server will however need some possible additional add-on tools with respect to enabling a Management GUI and Auto-Learning for the Mesh but I believe those are available today from some 3rd Parties on Github and my guess is sometime eventually from WireGuard itself.

Installation Guide

Step 1

First make sure LXD is installed in the Host/Server:

$ sudo snap install lxd

$ sudo lxd init # (configure LXD as you want)

Step 2

Make a copy of the “default” LXD container configuration Profile

$ lxc profile copy default ciabvpn

Step 3

Set the EDITOR environment variable to your favorite text editor. I use “nano”

$ export EDITOR=nano

Step 4

In this important step you are going to edit the *ciabvpn LXD configuration profile* and add a line which will tell LXD to load the Kernel’s “wireguard” module for you whenever an LXD container using ciabvpn profile is started.

This way every time one of the Tenant LXD containers runs, the WireGuard VPN will be available for use already to ensure secure communications with any other local/remote Hosts/Servers running that “tenants” LXD containers !

Edit the ciabvpn “profile”

$ lxc profile edit ciabvpn

to look like the following. Note ***the section in BOLD is what you need to add/change***

```
### This is a yaml representation of the profile.
### Any line starting with a '# will be ignored.
###
### A profile consists of a set of configuration items followed by a set of
### devices.
###
### An example would look like:
### name: onenic
### config:
###   raw.lxc: lxc.aa_profile=unconfined
### devices:
###   eth0:
###     nictype: bridged
###     parent: lxdbr0
###     type: nic
###
### Note that the name is shown but cannot be changed

config:
  linux.kernel_modules: wireguard
description: Default LXD profile
devices:
  eth0:
    name: eth0
    nictype: bridged
    parent: lxdbr0
    type: nic
  root:
    path: /
    pool: default
    type: disk
name: ciabvpn
used_by:
```

Save the changes and Exit the Editor.

Step 5

I*n all Host/Servers* install the wireguard PPA.

$ sudo add-apt-repository ppa:wireguard/wireguard

$ sudo apt upgrade -y

In all the Host/Servers also install a full wireguard and wireguard-utils:

$ sudo apt install wireguard wireguard-dkms wireguard-tools -y

Step 6

In all the Host/Servers create a "***tenant-base***" container that you can later use to clone real "tenant" containers from. Make sure to use "-p ciabvpn" to specify use of the ciabvpn "profile" you created in Step 4:

$ lxc launch ubuntu:b tenant-base -p ciabvpn

Step 7

In the LXD "tenant-base" Container only install wireguard-utils using:

$ lxc exec tenant-base bash

 # apt install wireguard-tools --no-install-recommends

Note:

You do NOT need to install WireGuard itself in the LXD Containers because they all "share" the Kernel of the Host/Server and the Host/Servers will have WireGuard installed into their Kernel(s) in Step 5.

Shutdown your new "tenant-base" container:

 # shutdown -h now

You should now be out of the LXD Container "tenant-base" and back on your Host/Server.

Step 8

From this point on in the Document whenever you are doing something reguarding configuration for a specific Tenant you will not use the "tenant-base" LXD container but will first clone the "tenant-base" container and use that new copy to work with.

Example: you have a new "tenant" so you want to name that "tenant" something like "*tenant-00001*"

$ lxc copy tenant-base tenant-00001

Then you can enter the container "*tenant-00001*" and configure it for that tenant's specific application needs as you would any Linux server (adopted to your own Tenant container naming scheme):

$ lxc exec tenant-00001 bash

NOTE:

Initial foundation configuration setup for CIAB Internetworking is now complete

Per-Tenant internetworking Solution Choices

Any Tenant may require compute/storage resources located in multiple locations. This may occur for many reasons such as:

1. Compute/storage resources required by the tenant are beyond what any single location has the capability to provide
2. Tenant is geographically dispersed and it is performant or cost beneficial to locate some compute/storage resources closer to those geographic locations/regions.
3. Pricing/Cost benefit of utilizing in-house vs Cloud -or- one Cloud vendor vs another Cloud vendor whether on a permanent -or- on a temporary basis.

   As LXD containers can be migrated between Hosts/Servers relatively easily the possibility of temporarily moving Tenant compute/storage resources because of pricing differentials has become a distinct possibility.

To facilitate such communications flexibility the chosen Solution must be flexible in its ability to enable rapid deployment or redeployment of Tenant compute/storage resources.

Current internetworking Solution choices are:

4. **O**pen **V**irtual **S**witch (**OVS**) and per-tenant **VXLAN** (**V**irtual e**X**tensible **LAN**) and **VNID**s (**V**irtual **N**etwork **Id**entifier) assigned on per-tenant basis.
5. Different **V**irtual **P**rivate **N**etwork (**VPN**) with VPN ID assigned on per-tenant basis.

Open Source

The Solution chosen will:

- LXD Hosts/Servers will be Ubuntu Linux using Long Term Support (LTS) releases
- Utilize LXD Container technology
- be Open Source licensed with source code available for review.
- Usually require integration of multiple open source technologies to satisfy Project KPI Goals.

Secure Communications

All Multi-Tenant LXD containers will by default be "***un-privileged***" Containers, meaning "root" in the Container is not "root" in the Host/Server.

Communications between LXD container resources on different LXD Hosts/Servers will be secured by accepted industry encryption algorithms and by the Solution chosen.

LXD Node/Server/Host Installation

Initial LXD installation on the multi-node Hosts/Servers follows standard practice:

$ sudo snap install lxd

$ sudo lxd init

Note: For "lxd init" accept/configure all answers as you normally would. You can also just accept all the "defaults"

Creating Per-Tenant LXD Bridges

The CIAB LXD Networking for Multi-Tenant, Multi-Node, Multi-Cloud Architecture uses the philosophy of each Tenant's LXD configuration requiring and utilizing an LXD bridge configured and dedicated on a per-tenant basis.

Normally, the installation of LXD would create a default bridge named LXDBR0.

In addition to the "lxdbr0" bridge created by "sudo lxd init" *it is recommended to adopt a naming convention that aids in also identifying which LXD bridge belongs to which Tenant and follow that naming convention on every multi-node/host/server you implement.*

Example:

tenant-00001-br

where "00001" is an identifier for tenant-00001

We will use the command "***`lxc network create`***" to create the CIAB Client bridges which will let LXD manage them in addition to the default "lxdbr0" bridge.

It makes creating the bridges *really* easy.

From the command usage:

$ lxc network create [<remote>:]<network> [key=value…]

Creates a network for LXD to use. So for our tenant-00001 an example bridge create command might be:

$ lxc network create tenant-00001-br bridge.driver=openvswitch

The above example command *would create a new **OpenVSwitch (OVS)** network bridge* with the name "tenant-00001-br".

Now one problem is that you have to specify this Bridge (re tenant-00001-br) somewhere so when you use LXD to create Containers for "tenant-00001" those Containers will all get access to and utilize the "tenant-00001-br" bridge.

LXD utilizes whar are called "**configuration profiles**" for this.

After LXD installation there is one profile called “default”. However, for networking purposes “default” only specifies the “lxdbr0” bridge.

For the CIAB multi-tenant, multi-node/server/host architecture though it would be very useful to create a new “configuration profile” dedicated specifically to each multi-tenant.

This will become especially time-saving when you deploy many containers, possibly on many “nodes” for each multi-tenant.

Tenant LXD Profiles

To facilitate creating an OpenVirtualSwitch (OVS) bridge dedicated to the LXD containers of each Tenant-xxxxx, especially when using a Multi-node architecture, it will be useful to create an LXD configuration “profile” for each Tenant-xxxxx.

> **NOTE:**
> The “**lxc profile edit**” command defaults to using Vi editor. If you want to use NANO then you should execute the following **before** using the “lxc profile edit” command:
>
> ***$ export EDITOR=nano***

To create individual Tenant-xxxxx LXD profiles we start by copying the ***default*** configuration Profile to a new configuration Profile. In this document we will name these ***tenant-xxxxx-profile*** where “xxxxx” is an identifier for a particular Tenant in a Multi-Tenant environment

Example: *note: you can name the new Configuration Profile whatever you want*:

```
$ lxc profile copy default tenant-00001-profile
```

After that, we need to edit the new **tenant-00001-profile** file, to make it have multiple network devices.

```
$ lxc profile edit tenant-00001-profile
```

You'll see something like this in the file:

```
config: {}
description: Default LXD profile
devices:
  eth0:
    name: eth0
    nictype: bridged
    parent: lxdbr0
    type: nic

root:
  path: /
  pool: default
  type: disk
```

Make a new blank line after that `type` line, and add lines, to make it look like this:

```
config: {}
```

```
description: Default LXD profile
devices:
  eth0:
    name: eth0
    nictype: bridged
    parent: lxdbr0
    type: nic
  eth1:
    nictype: bridged
    parent: tenant-00001-br
    type: nic
```

***** Don't edit anything else just save it UNLESS you know what you are doing*****

From this point forward any time you want to create a new LXD Container for tenant-00001 you have to tell LXD to create the new container using the ***`tenant-00001-profile`*** configuration p*rofile*:

Example:

```
$ lxc launch ubuntu:b CN1 -p tenant-00001-profile
```

This will tell LXD to use the "***`tenant-00001-profile`***" configuration profile we just set up as to create Container named "CN1".

CN1 will have access to both ***`lxdbr0`*** and ***`tenant-00001-br`*** on ***`eth0`*** and ***`eth1`*** respectively within the container.

NOTE: ***One major caveat:*** The system isn't smart, and doesn't configure `eth1` properly.

As such, you need to go into your Ubuntu system, and set up the configuration for `eth1` yourself, to set it to either DHCP or static IPs within the `tenant-00001-br` config's IP ranges. Otherwise, that interface won't ever be brought up.

As this configuration will vary from OS to OS, I can't give you a clear-cut answer here on how to configure each and every network interface on every image. There's other resources available for that.

Finally like above, launch your LXD container and specify the tenant-00001-profile for it.

$ lxc launch ubuntu:b tenant1 -p tenant-00001-profile

You won't need to do any specialized configuration for the network interface, as the default is to set the first interface to DHCP and that will auto-configure properly.

Example - Creation and Configuration of a Tenant Bridge and Profile

NOTE: The following example supposes a Tenant named "tenant-00001".

EDIT THE FOLLOWING TO CORRECT SHIT :-)

Step 1

Create the Tenant's Container network and network "profile" for use by LXD containers created later for that Tenant.

$ lxc network create tenant-00001-br

Step 2

Edit the *"**Network**" Configuration Profile* of tenant-00001-br

$ lxc network edit tenant-00001-br

so it looks similar to the following by using this command (*Note: your IP addresses will probably be different than these)*:

```
### This is a yaml representation of the network.
### Any line starting with a '# will be ignored.
###
### A network consists of a set of configuration items.
###
### An example would look like:
### name: lxdbr0
### config:
###   ipv4.address: 10.62.42.1/24
###   ipv4.nat: true
###   ipv6.address: fd00:56ad:9f7a:9800::1/64
###   ipv6.nat: true
### managed: true
### type: bridge
###
### Note that only the configuration can be changed.

config:
  ipv4.address: 10.140.45.1/24
  ipv4.nat: "true"
  ipv6.address: fd42:bfa2:8ac3:5431::1/64
  ipv6.nat: "true"
description: Tenant 00001
```

name: tenant-00001-br
type: bridge
used_by:
managed: true
status: Created
locations:
- none

NOTE 1: *in the above the IPv4 and IPv6 network addresses were randomly assigned by the LXD daemon software when the Network Profile was created. Every Tenant will be provided a different IPv4 and IPv6 network subnet, address and address range*

NOTE 2: *if you only want some "tenant" to have a smaller range of IP addresses used when creating "their" containers then you can do that by adding the following configuration statement to the "config:" section above:*

ipv4.dhcp.ranges: 10.140.45.200-10.140.45.220

Example *to limit the Tenant to just 20 IPv4 addresses & thus just 20 containers with IPv4 addresses ranging from 10.140.45.200 to 10.140.45.220:*

config:
ipv4.address: 10.140.45.1/24
ipv4.nat: "true"
ipv4.dhcp.ranges: 10.140.45.200-10.140.45.220
ipv6.address: fd42:bfa2:8ac3:5431::1/64
ipv6.nat: "true"

Step 3

Create a new container and specify to create it using the new "tenant-00001-br" profile.

$ lxc launch ubuntu:b tenant-1-cn1 -p tenant-000010br

The above command would create a new Ubuntu 18.04 container, name it "tenant-1-cn1 and use our special Tenant-00001-br profile (network and config profile) to create it.

After container "tenant-1-cn1" is created and started you can verify that it got the appropriate IP addresses by entering the container and running "ifconfig" to check how the Container's ETH0 interface is configured.

From your Host/Server you could also just use “lxc list tenant-1-cn1” to verify your desired/configured Tenant-1 IP addresses are as you expect.

$ lxc exec tenant-1-cn1 bash

root@tenant-1-cn1:~# **ifconfig**
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.140.45.156 netmask 255.255.255.0 broadcast 10.140.45.255
inet6 fe80::216:3eff:fe99:7219 prefixlen 64 scopeid 0x20<link>
inet6 fd42:bfa2:8ac3:5431:216:3eff:fe99:7219 prefixlen 64 scopeid 0x0<global>
ether 00:16:3e:99:72:19 txqueuelen 1000 (Ethernet)
RX packets 130 bytes 535415 (535.4 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 103 bytes 9255 (9.2 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 11 bytes 923 (923.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 11 bytes 923 (923.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

Various pieces of config info I plan to use and incorporate somewhere in the main Document

---

```
#!/bin/bash

# source:  http://www.panticz.de/lxd/network/openvswitch

# make sure openvswitch components are installed.

$ sudo apt install openvswitch-switch openvswitch-vtep -y

# create bridge

ovs-vsctl add-br tenant-00001-br

# ifconfig tenant-00001-br up
ip link set tenant-00001-br up

ovs-vsctl show

# connect ovs bridge to external network – assumes ETH0 interface
# change that if its not correct

ovs-vsctl add-port tenant-00001-br eth0

ifconfig eth0 0

dhclient tenant-00001-br -v

ip a show tenant-00001-br

route -n

# create LXD container
#
# NOTE - Change – change these to match SUDO LXD INIT  settings hat you would
# have used to create your containers

lxc profile create disk-only

lxc storage create pool1 dir

lxc profile device add disk-only root disk path=/ pool=pool1

lxc profile show disk-only
```

lxc launch ubuntu:18.04 ovs1 -p disk-only

lxc config device add ovs1 eth0 nic nictype=bridged parent=tenant-00001-br host_name=vport11

lxc launch ubuntu:18.04 ovs2 -p disk-only

lxc config device add ovs2 eth0 nic nictype=bridged parent=tenant-00001-br host_name=vport12

lxc network list

#!/bin/bash

source:
https://thomas-leister.de/en/container-overlay-network-openvswitch-linux/

make sure openvswitch is installed

sudo apt install openvswitch-switch openvswitch-vtep -y

for each of the Multi-tenants create an OVS bridge

sudo ovs-vsctl add-br tenant-00001-br

The ethernet MTU of the switch must be reduced because of all the
tunneling and IPsec/VLAN “overhead”

sudo ovs-vsctl set int tenant-00001-br mtu_request=1416

The cn-brX bridge on each host gets connected to the virtual switch.
A VLAN tag “200” (or whatever number you choose) is applied to separate
each Multi-Tenant container network from one another.
Execute something like this on ALL Multi-Nodes

 # NOTE – CHANGE - sudo ovs-vsctl add-port ovsbr0 cont-mgmnt0 tag=200

In the next step both host switches are connected to each other

NOTE – CHANGE - GRE to what it needs to be !

Host 1 (this is 1 line...)
ovs-vsctl add-port ovsbr0 gre0 -- set interface gre0 type=gre options:remote_ip=1.1.1.1
options:psk=mykey

Host 2 (this is 1 line...)

ovs-vsctl add-port ovsbr0 gre0 -- set interface gre0 type=gre options:remote_ip=2.2.2.2 options:psk=mykey

Every container for Multi Tenant-00001 is now able to connect to any other
container on the same tenant-00001-br network - no matter on which host the containers run
From 10.8.2.2 on Host 1 to 10.8.2.3 on Host 2
ping 10.8.2.3

SOURCE: https://openschoolsolutions.org/set-up-network-bridge-lxd/

NETPLAN

As of Ubuntu 18.04 Netplan is used to configure the network connections. The configuration files can be found under **/etc/netplan/**. A definition for the bridge could look like this:

```
$ cat /etc/netplan/50-cloud-init.yaml
# This file is generated from information provided by
# the datasource.  Changes to it will not persist across an instance.
# To disable cloud-init's network configuration capabilities, write a file
# /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg with the following:
# network: {config: disabled}
network:
    ethernets:
        enp3s0:
            dhcp4: no
    version: 2
    bridges:
        br0:
            dhcp4: no
            addresses:
            - 10.10.10.5/24
            gateway4: 10.10.10.254
            nameservers:
                addresses:
                - 10.10.10.254
            interfaces:
            - enp3s0
```

In the upper part you configure the real network card (*enp3s0*) and don't assign an address to it. Then the definition of the network bridge follows. It is set up like a static network connection and also contains the key *interfaces*. There you define which real network card should be "bridged". You will find further (more complex) examples of network bridges on the official website.

SOURCE for more NetPlan configs: https://netplan.io/examples#configuring-network-bridges

Now the following command applies the changes to the network settings:

$ netplan apply –debug

Assign Network Bridge

Once you have finished setting up the network bridge and it gets the correct IP address, you have to tell the LXD container to get its IP address from the network bridge. This can be done with the following command:

$ lxc config device add containername eth0 nic nictype=bridged parent=br0 name=eth0

With *name=eth0* you define under which name the network card can be found in the container. Now you can configure eth0 in the container as you like. From now on the container will get an IP address from the host network.

Conclusion

You can set up a simple network bridge quit easily and assign it to a container. This allows other users on the network to access a web application without the need to set up a reverse proxy on the container host. More complex scenarios are also possible (VLANs, multiple bridges to get containers into different networks, etc.), but this would go beyond the scope of this short article.

Source: http://www.panticz.de/index.php/openvswitch

Open vSwitch – Miscellaneous Configuration Commands/Functions

 # Install openvswitch

sudo apt install -y openvswitch-switch openvswitch-vtep

 # List

list interfaces

```
sudo openvswitch_vswitchd ovs-vsctl list-ifaces br-int

# show bridges

sudo openvswitch_vswitchd ovs-vsctl list-br

# show ports

sudo openvswitch_vswitchd ovs-vsctl list-ports br-int

# Create interfaces

# create bridge

sudo ovs-vsctl add-br mybridge

# ifconfig mybridge up

sudo ip link set mybridge up
sudo ovs-vsctl show
sudo ovs-dpctl show

# CREATE

echo sudo openvswitch_vswitchd ovs-vsctl -- --may-exist add-port br-int my-if1 -- \
set Interface my-if1 type=internal -- \
set Interface my-if1 external-ids:iface-status=active -- \
set Interface my-if1 external-ids:attached-mac=${CTL_HOST_MAC} -- \
set Interface my-if1 external-ids:iface-id=${PORT_ID} -- \
set Interface my-if1 external-ids:skip_cleanup=true

# Create Port

sudo openvswitch_vswitchd ovs-vsctl -- --may-exist add-port br-int my-port1 -- \
    set Interface o-hm0 type=internal -- \
    set Interface o-hm0 mac="${PORT_MAC}" -- \
    set Interface o-hm0 external-ids:iface-status=active -- \
    set Interface o-hm0 external-ids:iface-id=${PORT_ID} -- \
    set Interface o-hm0 external-ids:skip_cleanup=true -- \
    set Interface o-hm0 external-ids:attached-mac=${PORT_MAC}

# create VLAN

sudo ovs-vsctl add-port br0 vlan10 tag=10 -- set Interface vlan10 type=internal
sudo ip addr add 192.168.0.123/24 dev vlan10

# Show / List

ovs-vsctl show
```

```
ovs-vsctl list bridge
ovs-vsctl list port
ovs-vsctl list interface

# Create

sudo ovs-vsctl -- --may-exist add-port br-int o-hm0 -- \
set Interface o-hm0 type=internal -- \
set Interface o-hm0 external-ids:iface-status=active -- \
set Interface o-hm0 external-ids:attached-mac=${CTL_HOST_MAC} -- \
set Interface o-hm0 external-ids:iface-id=${PORT_ID} -- \
set Interface o-hm0 external-ids:skip_cleanup=true
```

 # DELETE

```
# delete port

sudo ovs-vsctl del-port br-int o-hm0

# delete ovs-tcpdump port

sudo ovs-vsctl del-port br-tun ovsmiXXXXXX

# delete bridge
ovs-vsctl del-br mybridge

# CLI
# http://www.sznote.net/?p=1032

sudo ovs-vsctl set int ovsbr0 mtu_request=1416

sudo ovs-appctl fdb/show mybridge
sudo ovs-ofctl show mybridge
sudo ovs-ofctl dump-flows mybridge

# fake interface for LXD?

sudo ovs-vsctl add-br br0
sudo ovs-vsctl add-port br0 eno1
sudo ovs-vsctl add-br vlan100 br0 100
lXc network attach-profile vlan100 default eth0
```

 # re TunTap devices

```
# tuntap devices

sudo ip tuntap add mode tap vport1
sudo ip tuntap add mode tap vport2
```

sudo ifconfig vport1 up
sudo ip link set vport2 up
sudo ifconfig
sudo ovs-vsctl add-port mybridge vport1
sudo ovs-vsctl add-port mybridge vport2
sudo ovs-vsctl show

get broken interface info

lxc exec os1-com2-dev --exec openvswitch_vswitchd ovs-vsctl show | grep -A2 qvo5b1aac7e-d4

delete port

lxc exec os1-com52-dev --exec openvswitch_vswitchd ovs-vsctl del-port qvo1c589412-d2

How do you create the LXD containers so they each get their own assigned VLAN?

You create them like you would create a normal Container using

$ lxc launch

command then use the

$ lxc config set command I have specified in the post

If you need to have many containers each within it’s own VLAN network.

Use OpenVSwitch in combination with LXD to achieve this.

There is no inherent facility in LXD to provide VLAN tag numbers to the interfaces.

So it is necessary to use a OpenVSwitch's “Fake bridge”.

You can read more about "Fake Bridge" in this article by Scott Lowe

VLANs with Open vSwitch Fake Bridges:
https://blog.scottlowe.org/2012/10/19/vlans-with-open-vswitch-fake-bridges/

Example:

You have OpenVSwitch bridge named vm-bridge

You want to add 10 fake bridges ranging from VLAN 20 to VLAN 30.

Here's a short Bash script to show how you can do it:

```
for i in $(seq 20 30); do
   ovs-vsctl add-br vlan$i vm-bridge $i
done
```

In LXD you can specify the bridge to which it will connect containers to.

So using another similar small Bash script you can create 10 containers using a similar loop and assign each new LXD Container to a specific VLAN.

Further to bind each container to the "fake bridge" this step is needed:

```
for i in $(seq 20 30); do
   lxc config device set CN$i eth0 parent vlan$i
done
```

BRIAN ->ADD THESE TO THE DOCUMENT SOMEWHERE…. !

How to create overlay networks using Linux Bridges and VXLANs

https://ilearnedhowto.wordpress.com/2017/02/16/how-to-create-overlay-networks-using-linux-bridges-and-vxlans/

and

Managing vxlan interfaces and networks with unicast in Linux

https://github.com/trueDGTL/manage_vxlan

Performance Tests

VM to VM thru VPN tunnel versus PING not thru tunnel

GloryTun using 100 Mbps = 40% slower over VPN (vpn=811msec 563 ping)

LXD PROFILES

References How To Configure & Use LXD Profiles

https://discuss.linuxcontainers.org/t/how-do-you-switch-profiles-on-a-container/2741

https://ilearnedhowto.wordpress.com/tag/vxlan/

https://blog.simos.info/how-to-add-both-a-private-and-public-network-to-lxd-using-cloud-init/

VXLAN and LXD – from Ron Kelley

For anyone else who might want to try VXLAN multicast between containers, here is a quick set of commands I used to get it working:

```
--------------------------------
ip route -4 add 239.0.0.1 eth1
ip link add vxlan1500 type vxlan group 239.0.0.1 dev eth1 dstport 0 id 1500
ifconfig vxlan1500 up
<edit LXD profile to match - set the nictype to “macvlan”, and the parent to
“vxlan1500”> >
--------------------------------

Simply replace the “vxlan1500” with your interface name of choice and pick you
physical ethernet port number (eth1 in the example above).  The parameters “id
1500” specify the VXLAN Network ID (0-16777215).

For what its worth, this is a huge win for me as I can setup a real environment
using software-defined VLANs w/out modifying any top-of-rack switches.  I simply
create a new VXLAN segment for each new customer on our LXD servers and deploy a
software firewall that manages traffic between the VXLAN segment with a local
gateway.
```