Virtual Lab – BSD Programming Workshop

BY ROLLER ANGEL

Our virtual lab will consist of a FreeBSD Host system that uses the FreeBSD Jails technology to provide each system we want to install within the lab its own separate environment to run services and perform its duties. These duties could be any number of things like serving up web pages, storing and retrieving database records, querying and answering DNS requests, caching system update files, etc. The idea is to build a solid foundation that will enable future growth for our virtual lab. Given that the nature of FreeBSD Jails is to provide a lightweight system for containing our services, we can rest assured knowing that any number of rabbit holes we may find ourselves in won't limit our creativity or exploration due to reaching our resource limitations. We can have a separate environment for each idea and not have to worry about the expense of provisioning another operating system to support the services necessary for that idea to flourish. I've experimented with many different methods of hosting operating system installs for my work and I'm pleasantly surprised by the peace of mind I get when provisioning a new jail. It's so economical! I'm no longer burdened with a financial concern and a question of how long this expense will be ongoing, rather it's just another environment in my ever growing lab and doesn't inherently come with a minimum monthly fee to use it. Not to get too technical and go down a financial rabbit hole regarding the cost of electricity, internet connectivity, host hardware; yes, I agree those things have a cost, but once they are in place, the addition of new hosts isn't anywhere near the considerations and expenses that go into the initial lab setup. I recommend repurposing an existing machine as the host machine, maybe even a laptop, as it comes with a built in battery backup, giving you time to gracefully shutdown your systems in the event of a sustained power outage. The issue of needing multiple physical network interfaces to connect physical Ethernet switches to Ethernet cables and access points that your physical hosts will use are not a material concern in our virtual lab. These interfaces can be created with words in a text file and virtual Ethernet cables can be created to connect the pieces of our virtual network.

FreeBSD Host

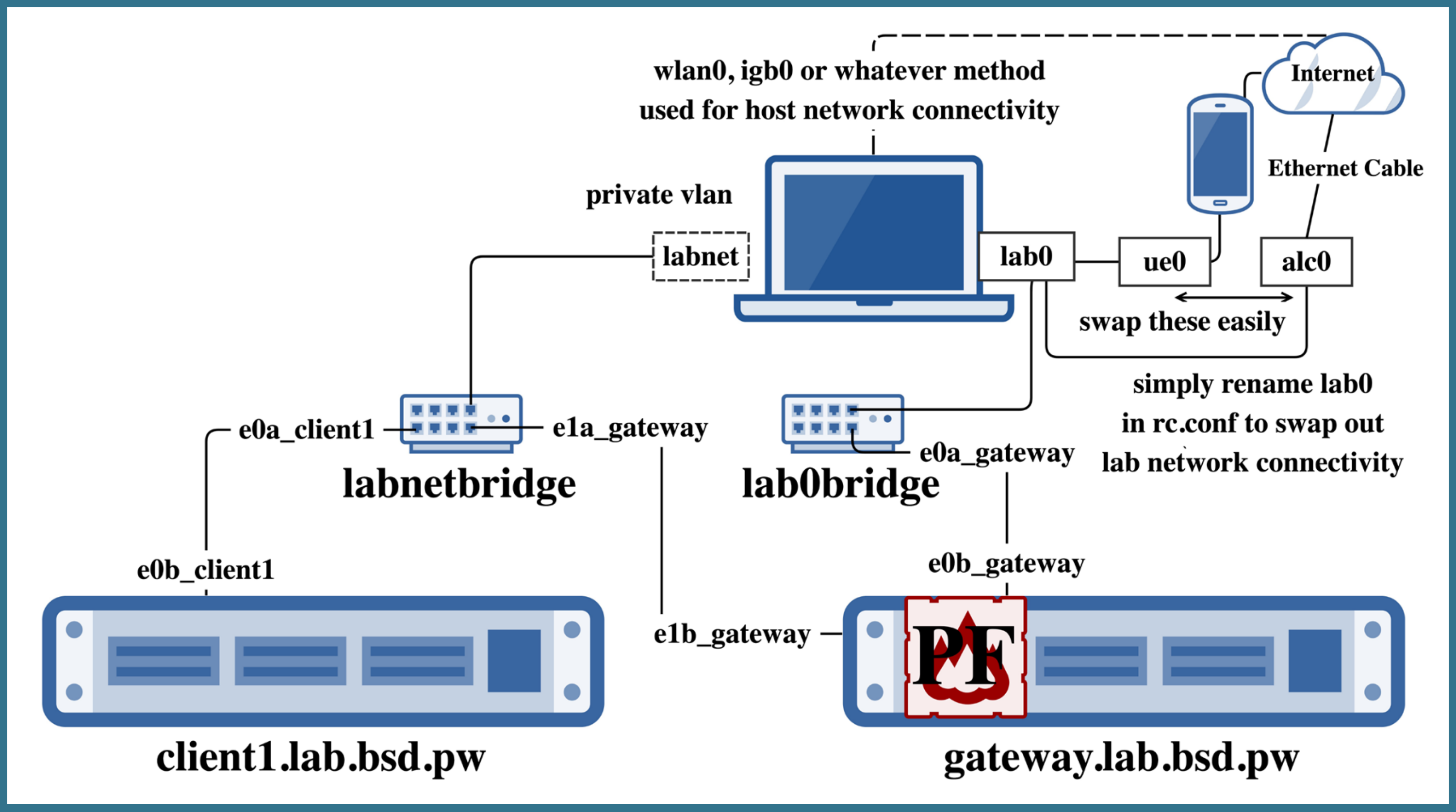
This host machine will need a few network interfaces. We want the host to have its own way out to the internet. This can be a good ole DHCP assigned address provided by a local router, or your host could be performing the role of router and have a direct connection to the outside world through a modem. However your host gets its internet connection, we'll want to have an additional network interface that we can use solely for our lab network. This interface may have a name like `em0`, `igb0`, or similar depending on the network card driver and the number of installed

```
Welcome to FreeBSD
1. Boot Multi user [Enter]
2. Boot Single user
3. Escape to loader prompt
4. Reboot
5. Cons: Video

Options:
6. Kernel: default/kernel (1 of 1)
7. Boot Options

Autoboot in 2 seconds. [Space] to pause
```

interfaces. Even if we don't have multiple physical network interfaces, we can always make a separate network interface to use with `cloned_interfaces`. In my case, I'm repurposing some laptops. One has a working WiFi card I use for connecting to the internet. This is an older Asus ROG laptop from the early 2010s. It has a built in Ethernet port as well. On my X1 Carbon 7th Gen ThinkPad, there aren't built in Ethernet ports. I just use the USB port and my Android USB Tethering. However, I'm most excited about the Dell Precision laptop. It's a great lab host as it comes with an Intel Xeon processor and can handle 128GB of RAM and multiple NVME hard drives. It has a built in ethernet port on the back of the laptop and the second one is a USB-C dongle that has an Intel `igb0` ethernet port on it that I use for the dedicated interface for the lab network. Finally, old towers are also great as a host, there are some fantastic PCI network cards you can install with mulitple Ethernet ports. The key here is to find what works for you, install FreeBSD and get going with our virtual lab. See the Getting Started With FreeBSD Workshop from the July/August 2022 *FreeBSD Journal* issue if you'd like to get some tips on installing and configuring FreeBSD.

Virtual Network Design

Just like in the physical server world where we have multiple physical Ethernet ports on our router/firewall systems, we can provision multiple interfaces for our lab router/firewall system which I will refer to as the gateway from now on. These interfaces will be connected to each other via bridges. Think of an Ethernet switch from a physical network, a bridge provides similar functionality. And for our Ethernet cables, we will use epair(4) interfaces. These consist of two sides, side A and side B. We'll hook up side A to the bridge and side B to our jail. This way all the jails can communicate with each other over the virtual bridge much like physical hosts in a network connected via Ethernet cable to a switch would be able to communicate with one another. The gateway jail will have an additional virtual cable to connect to a separate bridge that allows connectivity outside the lab and onto the internet. All jails will set their default route to point to this gateway. The way we give that separate bridge the ability to connect to the outside world is by assigning a physical interface as a member of the bridge. Again, this is akin to plugging the Ethernet cable into the switch, but in this case the Ethernet cable is plugged into the FreeBSD Host system and is then virtually plugged into the bridge. By doing this, the other virtual Ethernet cables that are also members of the same bridge will be able to communicate with it and get their packets flowing in and out of the lab environment.



Let's break it down, each jail gets its own network using a technology called vnet. We attach our physical interface on the FreeBSD Host to one virtual bridge and we create a second virtual bridge for our lab network that has only virtual interfaces from lab jails connected to it. We then use routing and firewall rules to push packets around as we see fit.

Firewall

We'll be using PF for the firewall. The way FreeBSD Jails works is that each jail is sharing the host kernel, so if there's a kernel module that you want to have access to inside the jail, you just need to allow it and then configure access in the `jail.conf` settings for the particular jail. Take a look at the `/etc/defaults/devfs.rules` file. For our gateway to use the PF firewall, we'll need to set the configuration of the jail to use the pf ruleset listed in this file. We'll setup our own custom devfs rules later on and include the configuration from the `devfsrules_jail_vnet` rule.

Configuration

Now that we've covered what we're going to do, let's get to doing it. We start with a FreeBSD Host running 13.1-RELEASE. As described above, this host should have an active internet connection. We'll use that connection to download some files for use in creating our jails. The host has NTPD running, so it gets accurate time. Check for any services listening on the host with `sockstat -46` and turn them off if unused. Remember that the host should be limited in what it does—we'll have plenty of fun things to do in jails inside our lab, so do your best to limit the services on the host. I plan on doing any management of my host in person by logging in with the attached keyboard and screen so I've not enabled SSH on the host.

Now we're ready to enable jails. A simple `sysrc jail_enable=YES` will do the trick. No need to install any package, jail management is built into FreeBSD. Take a look at the README file in `/usr/share/examples/jails` for some examples of how you might configure your jails. As you will see, there are many ways to go about jail configuration. I've done my research and picked the configuration method you'll see here. You're welcome to give one of the other approaches a try and see what fits. If you do go about this task another way, please consider writing about it so others can see what you've found useful and give it a try themselves. At this point, we've verified our host machine is ready for hosting jails and have enabled the jail service so we can do a quick `reboot` double check that minimal services are listening on the host and move on to creating the base configuration for all our jails to use. When editing configuration files we'll be using `vim`, for basic editing tasks you really only need to know a handful of things, spend a few minutes going through the interactive exercises as part of the command `vimtutor` to get your bearings and you'll be a vim novice in no-time at all.

Note: We're running all the following commands as the root user. Type `sudo -i` to become root.

edit jail.conf

```
vim /etc/jail.conf
```

put the following into /etc/jail.conf

```
$labdir="/lab";
$domain="lab.bsd.pw";
path="$labdir/$name";
host.hostname="$name.$domain";
exec.clean;
exec.start="sh /etc/rc";
exec.stop="sh /etc/rc.shutdown";
exec.timeout=90;
stop.timeout=30;
mount.devfs;
exec.consolelog="/var/tmp/${host.hostname}";
```

base.txz

```
mkdir -p /lab/media/13.1-RELEASE
cd /lab/media/13.1-RELEASE
fetch http://ftp.freebsd.org/pub/FreeBSD/releases/amd64/13.1-RELEASE/base.txz
```

Gateway Jail

```
mkdir /lab/gateway
tar -xpf /lab/media/13.1-RELEASE/base.txz -C /lab/gateway
```

edit jail.conf

```
vim /etc/jail.conf
```

add to the bottom of the file

```
gateway {
  ip4=inherit;
}
```

feel free to add a user account to the jail with the following optional command, for this article we're just going to be using the user root

```
chroot /lab/gateway adduser
```

set the root password for the jail

```
chroot /lab/gateway passwd root
```

setup DNS resolution using OpenDNS servers

```
vim /lab/gateway/etc/resolv.conf
```

add the following lines to resolv.conf

```
nameserver 208.67.222.222
nameserver 208.67.220.220
```

copy the hosts time zone setting

```
cp /etc/localtime /lab/gateway/etc/
```

create an empty file system table

```
touch /lab/gateway/etc/fstab
```

start jail

```
jail -vc gateway
```

login to jail

```
jexec -l gateway login -f root
```

logout of the jail

```
logout
```

list jails

```
jls
```

stop the jail

```
jail -vr gateway
```

create devfs.rules

```
vim /etc/devfs.rules
```

add the following lines to devfs.rules

```
[devfsrules_jail_gateway=666]
add include $devfsrules_jail_vnet
add path 'bpf*' unhide
```

restart devfs

```
service devfs restart
```

verify devfs rules

```
devfs rule showsets
```

assign our new ruleset to the gateway jail

```
vim /etc/jail.conf
```

add the following line to the gateway { } config block

```
devfs_ruleset=666;
```

restart the gateway jail

```
service jail restart gateway
```

verify ruleset was applied to gateway jail

```
jls -j gateway devfs_ruleset
```

we expect to see 666 as the output of the above command

load the PF kernel module on host

```
sysrc -f /boot/loader.conf pf_load=YES
kldload pf
```

enable PF on gateway jail

```
sysrc -j gateway pf_enable=YES
```

edit pf.conf on gateway jail

```
vim /lab/gateway/etc/pf.conf
```

add the following config

```
ext_if = "e0b_gateway"
int_if = "e1b_gateway"
table <rfc1918> const { 192.168.0.0/16, 172.16.0.0/12, 10.0.0.0/8 }

#Allow anything on loopback
set skip on lo0

#Scrub all incoming traffic
scrub in
no nat on $ext_if from $int_if:network to <rfc1918>

#NAT outgoing traffic
nat on $ext_if inet from $int_if:network to any -> ($ext_if:0)

#Reject anything with spoofed addresses
antispoof quick for { $int_if, lo0 } inet

#Default to blocking incoming traffic, but allowing outgoing traffic
block all
pass out all

#Allow LAN to access the rest of the world
pass in on $int_if from any to any
block in on $int_if from any to self

#Allow LAN to ping us
pass in on $int_if inet proto icmp to self icmp-type echoreq
```

Configuring the Virtual Network

Setup an interface. Here we're using the dedicated network card named `alc0` and assigning it a name of lab0. All further configuration will use the interface name lab0 and the actual physical device it's assigned to can be changed by editing one line of configuration. This way it's easy to switch our interface if we move our lab to another host or add a new network interface down the road with a different name such as `igb0`, `re0`, or `em0`.

```
sysrc ifconfig_alc0_name=lab0
sysrc ifconfig_lab0=up
service netif restart
```

copy the Jail Interface Bridge automation script into our lab scripts directory and make it executable

```
mkdir /lab/scripts
cp /usr/share/examples/jails/jib /lab/scripts/
chmod +x /lab/scripts/jib
```

edit jail.conf

```
vim /etc/jail.conf
```

Gateway jail.conf

At this point, we're ready to move from inheriting the ip4 network from the host and use vnet, remove the gateway {} configuration block from /etc/jail.conf and replace it with the following

```
gateway {
  vnet;
  vnet.interface=e0b_$name, e1b_$name;
  exec.prestart+="/lab/scripts/jib addm $name lab0 labnet";
  exec.poststop+="/lab/scripts/jib destroy $name";
  devfs_ruleset=666;
}
```

create the internal LAN network for the jails in the lab

```
sysrc cloned_interfaces=vlan2
sysrc ifconfig_vlan2_name=labnet
sysrc ifconfig_labnet=up
service netif restart
```

destroy and recreate gateway

```
jail -vr gateway
jail -vc gateway
```

configure networking for gateway jail

```
sysrc -j gateway gateway_enable=YES
sysrc -j gateway ifconfig_e0b_gateway=SYNCDHCP
sysrc -j gateway ifconfig_e1b_gateway="inet 10.66.6.1/24"
service jail restart gateway
jexec -l gateway login -f root
```

test connectivity

```
host bsd.pw
ping -c 3 bsd.pw
```

exit the jail

```
logout
```

create another jail that only has one interface that's attached to the labnet LAN network

```
vim /etc/jail.conf
```

add the following to the bottom of the file

```
client1 {
  vnet;
  vnet.interface="e0b_$name";
  exec.prestart+="/lab/scripts/jib addm $name labnet";
  exec.poststop+="/lab/scripts/jib destroy $name";
  devfs_ruleset=4;
  depend="gateway";
}
```

make the directory structure for the new jail

```
mkdir /lab/client1
tar -xpf /lab/media/13.1-RELEASE/base.txz -C /lab/client1
```

set the root password

```
chroot /lab/client1 passwd root
```

setup DNS resolution using OpenDNS servers

```
vim /lab/client1/etc/resolv.conf
```

add the following lines to resolv.conf

```
nameserver 208.67.222.222
nameserver 208.67.220.220
```

copy the hosts time zone setting

```
cp /etc/localtime /lab/client1/etc/
```

create an empty file system table

```
touch /lab/client1/etc/fstab
```

start jail

```
jail -vc client1
```

configure networking for client1 jail

```
sysrc -j client1 ifconfig_e0b_client1="inet 10.66.6.2/24"
sysrc -j client1 defaultrouter="10.66.6.1"
service jail restart client1
```

login to jail

```
jexec -l client1 login -f root
```

test connectivity

```
host bsd.pw
ping -c 3 bsd.pw
ping -c 3 10.66.6.1
```

grab a sample tcsh profile

```
fetch -o .tcshrc http://bsd.pw/config/tcshrc
chsh -s tcsh
```

exit the jail

```
logout
```

The next time you login, you'll have a green prompt due to the tcshrc settings, enjoy! Now you have a virtual lab with it's own virtual network that has a physical interface reserved for the outbound connection to the internet. Since we named the interface lab0, we can easily update it. Go ahead and give that a try. For instance, you can plug in an Android phone that has an internet connection, WiFi or Cellular is fine, as either will do. Go to the network settings on the phone after you plug it into the host and enable USB Tethering. A ue0 interface will now be available for use. Update the line in **`/etc/rc.conf`** that says **`ifconfig_alc0_name="lab0"`** to be **`ifconfig_ue0_name="lab0"`**. Reboot. Login to either jail and test connectivity. Your network has been swapped out. Your lab is now mobile!

I hope you had fun following along with this article. I'm super passionate about FreeBSD and sharing what I learn with others brings me joy. I hope you do amazing things with FreeBSD in your lab and I look forward to chatting with many of you at one of the fantastic BSD conferences.

ROLLER ANGEL spends most of his time helping people learn how to accomplish their goals using technology. He's an avid FreeBSD Systems Administrator and Pythonista who enjoys learning amazing things that can be done with Open Source technology — especially FreeBSD and Python — to solve issues. He's a firm believer that people can learn anything they wish to set their minds to. Roller is always seeking creative solutions to problems and enjoys a good challenge. He's driven and motivated to learn, explore new ideas, and to keep his skills sharp. He enjoys participating in the research community and sharing his ideas.