SEE TEXT ONLY

GETTING STARTED WITH iohyve

By Trent Thompson

If you've ever had to navigate the system administration waters, you've probably come across virtualization as a way to make your life a little easier. Instead of having racks full of computer hardware, virtualization allows you to essentially turn one computer into many computers, thus saving precious space and computing cycles. Virtualization can help with small tasks, such as installing VirtualBox or VMWare on a workstation to test drive the new hottest Linux distribution. The tech giant Amazon uses Xen virtualization to power its Amazon Web Services cloud product, which made them over $12 billion in 2016.

Of course, the purpose of this article isn't to make you a billionaire but to help make your life a little easier. We're going to do this by leveraging two key technologies built into FreeBSD: ZFS and bhyve.

GETTING STARTED WITH iohyve

iohyve's Origin

ZFS is the filesystem and logical volume manager that has been in FreeBSD since FreeBSD 7 in 2008, when it was still considered experimental. It originated at Sun Microsystems (now Oracle) in 2005, when it was released with the OpenSolaris operating system, and since then has been ported to many other operating systems. Key features in ZFS include automatic data integrity, software RAID (called RAID-Z), creating logical volumes that act as raw disks (Zvols), and snapshots (and the ability to roll back to previous snapshots). We'll get into why these features are important to iohyve later.

Of course, we also need some sort of virtualization technology to help turn one computer into many computers. Indeed, FreeBSD has had jails since about FreeBSD 4 (longer than ZFS support). FreeBSD jails provide a lightweight way to virtualize the FreeBSD userland, meaning you can only virtualize FreeBSD, as all of your jails will share the same FreeBSD kernel. It's a great way to turn one FreeBSD host into many FreeBSD "virtual machines," each with their own IP addresses and separation from each other (one jail cannot access data in another jail by default, and by design). Solaris has a similar technology called Solaris Zones, which is still used today by virtualization solutions like SmartOS (based on OpenSolaris). Linux also has its own containers, provided by LXC and even Docker. The problem with most of these solutions is that they cannot virtualize other operating systems. For instance, you can't run Linux in a jail, or FreeBSD in a Docker container.

This is where hardware-assisted virtualization comes in to help. Using CPU technologies like VT-x on Intel CPUs or AMD-V on AMD CPUs, a special software suite called a hypervisor allows one operating system to virtualize another. Note that I didn't use the word "emulate." An emulator, such as QEMU, only emulates a processor, while a hypervisor connects a virtual machine to the CPU itself, thus having huge performance increases over emulation. There are a few different solutions that I've mentioned, including Xen. Xen is a powerful hypervisor, and is even available on FreeBSD. Since its beginnings around 2003, it has grown considerably. Linux distributions have a built-in hypervisor called KVM (Kernel-based Virtual Machine). Linux KVM has been around since 2007, and is even used by SmartOS to virtualize other operating systems where Zones won't cut it. The hypervisor that iohyve uses is bhyve, which has been built into FreeBSD since FreeBSD 10 in 2014. You'll note that this is considerably young in comparison to the other hypervisors mentioned. The bhyve hypervisor is a much more lightweight hypervisor, with its kernel module and userland utilities taking up less than 500k of space. The bhyve hypervisor has also been ported to other operating systems including Mac OS X where it is called xhyve, and is used by Docker to do the heavy lifting for their Linux containers.

The bhyve hypervisor may be lightweight, but it is a powerful tool with many features. A few years ago while navigating my own system administration waters, I needed to virtualize some Linux servers. After trying many different solutions, I landed on FreeBSD, mainly for the ZFS support. At the time, I was using Oracle VirtualBox with phpVirtualBox as a GUI to remotely administer virtual machines. I configured VirtualBox to store all the virtual machines on a single ZFS dataset. This worked okay, as I would be able to make snapshots of the dataset to make backups, but I wasn't able to store virtual machines on separate ZFS datasets, which, to me, made more sense. That way I can make snapshots of individual virtual machines and not a snapshot of all the virtual machines. For a very short time, I tried to modify VirtualBox to create a new dataset to put a virtual machine into, but this turned out to be more work than I wanted to put in. Around that same time, I was playing with FreeBSD jails, and was even using them to separate phpVirtualBox from the VirtualBox host so if someone exploited the PHP webserver, they would be stuck in the jail, and not on the VirtuaBox host itself. At first, I used ezjail-admin to implement this, but quickly dropped it for iocage, a jail manager that utilized ZFS in a way that was similar to what I wanted to do with VirtualBox (each jail is on its own ZFS dataset).

As previously mentioned, jails are wonderful, but they can only "virtualize" FreeBSD, not other operating systems like Linux. I really loved the idea of iocage though. Not only did each jail have its own dataset, but it also stored the properties of each jail in the ZFS user properties of the dataset, essentially eliminating the need for configuration files or a dependency on a database. I was also attracted to the iocage project because they used shell scripting, meaning that the code was easily understood at a glance, and didn't require any compilation. The iocage project has since moved on from these two attractions, mainly due to its enormous growth since then, where they have opted for UCL configuration to speed things up (ZFS user properties queries can be very slow on a busy system) and have switched to Python to alleviate the headaches of writing a massive shell script. At first, my idea was to add bhyve support to iocage, but it quickly became clear that this was like putting a square peg in a round hole, while the size of the round hole was constantly being changed (with iocage growing at such a fast rate, it was hard to keep up with the changes).

Eventually one afternoon, I hacked out a basic script that mimicked the simple command line interface of iocage to mate bhyve and ZFS together. In the beginning, it consisted of just one script, and had very limited abilities like only support for FreeBSD virtual machines, and support for only one virtual disk that was stored as a file. I uploaded the script as a text file to GitHub's Gist repository and showed it to a handful of people. They all wanted more, and so I eventually created an actual GitHub repository, which is still around. This allowed me to turn a hacked-out script into a full-fledged project, with a wiki that can be used like a handbook, an issue tracker to help with bug reporting, and the ability for anyone to help contribute with pull requests. Since its creation, iohyve has grown with more and more features and bug fixes being added along the way. One defining moment in iohyve history was the suggestion to use ZFS ZVOLs as a way to better utilize ZFS. ZVOLs can be created and snapshotted just like regular datasets; however, they appear to the operating system as basically disk devices. This is very similar to the way many Linux KVM and Xen solutions store their virtual machines on LVM partitions, cutting some performance overhead. Eventually, the ability to also run the other BSDs, like NetBSD and OpenBSD, as well as various Linux distributions, had support in iohyve with the addition of grub2-bhyve support, which essentially acts like the GRUB bootloader for bhyve (as the name implies). This is because bhyve doesn't have a built-in BIOS to load the important bits of the operating system. There was bhyveload which will load the FreeBSD-based operating systems, and grub-bhyve, to help boot operating systems that can use GRUB to boot. Eventually, bhyve gained support for UEFI booting, meaning there was no longer a need to use bhyveload or grub-bhyve, as the UEFI firmware could be used to load the operating system, just like many modern computers use to load operating systems today on "bare metal." With this UEFI firmware, you can run many different types of operating systems, including modern versions of the Microsoft Windows operating system family. Another recent feature addition to bhyve was the ability to have a graphical console provided by UEFI-GOP. Originally, bhyve only had support for an emulated serial console, meaning the virtual machines didn't have support for a GUI with a mouse and keyboard, like Oracle VirtualBox's remote display feature. Since FreeBSD 11, bhyve has come built in with this UEFI-GOP support as a basic VNC server. Since BSDCan 2016, iohyve has had the ability to utilize this VNC server to help install operating systems that have graphical installers like CentOS and Windows. Although that's a cool feature, we won't get much into UEFI-booted bhyve virtual machines.

How to Use iohyve

So enough of the boring iohyve origin story. Let's learn how to actually use iohyve! For the purposes of this tutorial, I will be using FreeBSD 11 Release. Although most of the hottest new features (and feature branches) of bhyve are done on the current branch, I chose to go with a release simply for the ability to keep updates easy with freebsd-update and pkg. Your mileage may vary. An important part of iohyve, ZFS, can be installed and set up during the installation process. Even though you may want to keep your iohyve virtual machines, or "guests," on a separate zpool, I still chose to go with FreeBSD installed on ZFS. If you have the overhead to run virtual machines, you have the overhead to run ZFS (terms and conditions may apply). I often chose to lump all the disks I have in a box (even if it's just one) into a zpool and install FreeBSD onto it (usually as zroot). Once the installation is done, you are given the opportunity to drop to a shell to make any changes before rebooting into your new and fresh FreeBSD installation. I take this opportunity to install some dependencies and utilities to help with making the new installation ready to go. Your "stack" may be different, but this is just one I find useful to get started. Here's the one-liner that uses pkg:

```
pkg install sudo nano tmux git htop bhyve-firmware grub2-bhyve
```

The use of `sudo` is pretty straightforward: bhyve requires root (for now) and I don't trust users. Instead of giving them access to the root account, I delegate this through `sudo`. I know some of you are giving me weird looks with the use of nano, but it's what I've been using to edit config files since I've been editing config files. Sure, edit in base is simpler than vi, but nano's interface is more muscle memory than thinking at this point. Next on the list is tmux, the terminal multiplexor. This is handy for a number of reasons, the first of which is: you can keep your sessions opened even if your SSH session dies. Since bhyve doesn't utilize a graphical console by default, all communication can be done over null modems (like a serial connection). The use of tmux allows for you to open a new tmux window or pane for each new console to an iohyve guest. I'll go into how it can be used to help monitor your iohyve host. I like to use git because even if you can install iohyve via ports or pkg, you can make sure you're getting the latest and greatest bug fixes from the iohyve GitHub master branch. I also install htop honestly to get the cool CPU graph output, for better resource monitoring at a glance. The bhyve-firmware package installs the bhyve UEFI firmware. We won't be going into that, but it's good to have on hand just in case. You can find more info on iohyve and UEFI in the man page. Lastly is grub2-bhyve, which allows us to run other BSDs or Linux distributions as iohyve guests.

After everything is installed, I add my default user to the sudoers file and reboot into the new installation. From here on out, I use SSH to connect to the machine. In theory, this can all be done via the console as well. The first thing I do after logging in is start a new tmux session with simply:

```
tmux
```

I should warn you, if you've never used tmux before, it's awesome, and is kind of like GNU screen. Next, I install a new copy of iohyve from GitHub with:

```
git clone
https://github.com/pr1ntf/iohyve.git
```

You can also fetch the latest master ZIP file if you look for it. There are also releases available, that are also available as a port and package. Now we can install it (if we are using sudo) with:

```
cd iohyve/
sudo make install clean
cd ~
```

Voila! The magic of makefiles moves everything where it needs to be, including a man page entry and an RC script. Now we need to set up iohyve itself. One setup command only needs to be run once; the other needs to be run anytime you start your iohyve host. This can be easily done with the RC script, but we'll get to that shortly. First, we set up iohyve on a zpool. In the example, I use the built-in zroot pool. You can use whatever pool you've set up.

```
sudo iohyve setup pool=zroot
```

Next, we want to set up required kernel modules and networking for our iohyve host. You can do this manually yourself, but if your iohyve host is just going to be used for iohyve, I suggest using this method. Note this method doesn't work with wireless. There is some documentation out there on using iohyve with wireless, but for our purposes, we are going to set up networking attached to an Ethernet device. All of your iohyve guests will be attached to a bridge, and that bridge is attached to an interface. There are more complicated setups than this, but those features are new to iohyve, and bugs are still being worked out as of this writing. Check the man page for more info. In the following example, I am going to use the em0 interface on my iohyve host. You can see on which interfaces you have working networking by a simple ifconfig.

```
sudo iohyve setup kmod=1 net=em0
```

You can make sure iohyve does this on every boot by adding the following to your rc.conf:

```
iohyve_enable="YES"
iohyve_flags="kmod=1 net=em0"
```

Next, we are going to set up our first iohyve guest. We'll create the guest, then change some properties of the guest so that we can run Ubuntu Linux. Next, we'll fetch an Ubuntu ISO directly to iohyve's local ISO repository. Any installation ISO needs to be in the iohyve repository, and needs to be added to the repository by iohyve itself (you cannot simply move an ISO to a directory).
First the creation, in this case I want to name it ubantuserver and I want the guest to have a 16GB virtual hard drive:

```
sudo iohyve create ubantuserver 16GB
```

Next we need to change some properties. I'm going to give the guest a description, change the RAM and CPU to give it more resources (2 Gig of ram and two virtual CPUs), and configure it to utilize grub-bhyve to boot the Ubuntu kernel. We can do this in one simple iohyve command:

```
sudo iohyve set ubantuserver description="Ubuntu 16.04 Server" ram=2048M cpu=2
loader=grub-bhyve os=Ubuntu
```

You can check to see if everything is set up properly with:

```
iohyve info -v
```

Be sure to put your description string between two double quotes. You don't have to set a description if you don't want to; the default description is the timestamp of when the guest was created. Next, we are going to fetch the ISO from Canonical. Since we aren't going to be using a GUI to install, we will grab the server edition from my fastest mirror (your mirror may vary, see the Ubuntu website for more info):

```
sudo iohyve fetchiso
http://mirror.pnl.gov/releases/xenial/ubuntu-16.04.2-server-amd64.iso
```

This will automatically fetch the ISO and put it where it needs to be. If you have already fetched the ISO, you can run something like:

```
sudo iohyve cpiso /full/path/to/iso.iso
```

Now here's the part where `tmux` becomes really useful. We're going to create a `tmux` window by typing Ctrl+B then "c". Ctrl+B is the default action key, similar to Ctrl+A in screen. The "c" creates the new window, pressing Ctrl+B and then "n" will cycle to the next window, and Ctrl+B and then "p" brings you to the previous window. In our new `tmux` window we are going to open a console to the new guest:

```
sudo iohyve console ubantuserver
```

Nothing should be there yet, because we haven't started the guest yet. Let's go over to our previous `tmux` window using the key command described above (Ctrl+B then C). We can see a list of ISOs that are in the local iohyve repository with:

```
iohyve isolist
```

There we should see the ISO we fetched:

```
ubuntu-16.04.2-server-amd64.iso.
```

So to begin our installation, we run the following command to boot up the new guest with the attached Ubuntu ISO:

```
sudo iohyve install ubantuserver
```

Now move on to your `tmux` window with the console open and if everything worked, you should see a GRUB menu if you're fast enough, or a wall of text scrolling by. If you have a GRUB prompt that looks like "`grub>`" then double-check your os property settings on the guest. Different distributions have different quirks that iohyve can work with. Your installation should go like any other Ubuntu installation, just be sure to choose the LVM install (which is the default). If you choose to install directly onto ext4 without LVM, you may want to set your os property to "debian" as the quirks should be similar to non-LVM Ubuntu installs. Sometimes you can get kernel messages in your console while installing, specifically during partitioning. This is fine, and just a product of using a serial console. If everything goes well, you should be prompted to reboot. Click okay and then head back over to your previous `tmux` window. If iohyve won't automatically reboot the guest, you can make sure it is no longer running and then start the guest for its first full boot:

```
iohyve list
sudo iohyve start ubantuserver
```

On your `tmux` window with the console open, you should start to see the new Ubuntu Guest boot up. From there you can do whatever it is you want to do with Ubuntu. If you don't like Ubuntu, take a peek at the man page to see what OSes work with iohyve's ability to manage quirks. You can rinse and repeat as long as you have the resources to do so. To keep an eye on my resources, I create a special `tmux` window with four different panes. Check the `tmux` man page for how to create and size panes, but if you are lazy like I sometimes am, you can just run these four commands in a different `tmux` window. Using panes just gives it a fancy "control panel" feel to it. First we run `htop` as root, and filter it to only view processes that start with `"bhyve"`. This will give us visibility into what bhyve guests are using how much CPU and RAM (with those fancy bar graphs!). In the next `tmux` window or pane, I like to have a current view of the installed guests and what resources they have, and whether or not they are running with iohyve info. The next two are systat outputs that give both a view of disk usage and network usage. Remember to correlate the `"tap"` interfaces in the systat output to the output of iohyve info.

```
sudo htop
iohyve info -sv
systat -iostat
systat -ifstat
```

Hopefully you now have the knowledge to maybe start moving some of those Linux servers lying around to a virtual host, or maybe just to have a nice sandbox to test new things in to help you learn a new skill. You may find that you like to monitor your resources differently, or that you only need to run FreeBSD guests, or only Windows guests. I hope that iohyve helps you navigate the system administration waters. If you have problems or questions with iohyve, or you'd like to request a new feature, or even contribute to iohyve, head on over to the GitHub page (https://github.com/pr1ntf/iohyve) and someone will eventually get to you. The iohyve project is run by volunteers, so don't expect an instant response, but we'll generally respond to a GitHub issue (https://github.com/pr1ntf/iohyve/issues) eventually.

TRENT THOMPSON is a security engineer by day and a FreeBSD and virtualization hobbyist by night, maintaining and contributing to The iohyve Project. When not doing BSD-related activities, you can find him tinkering with something else technical around the house, like musical synthesizers, model rockets, or micro-computers from the 1980s. You can never have too many hobbies.

DELL
BUILD SOMETHING GREATER.
Dell is now part of the Dell Technologies family. So you can make your mark in everything from storage and big data to the Internet of Things.
Apply at Dell.com/careers