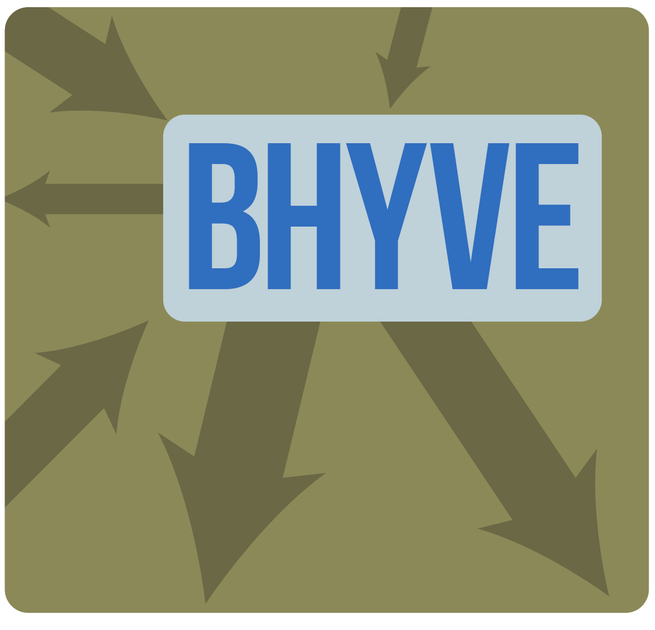

FOR THE LINUX AND WINDOWS USERS

BY JASON TUBNOR

The FreeBSD bhyve hypervisor was announced to the world in May 2011 by Neel Natu and Peter Grehan and then gifted to FreeBSD from NetApp. This finally gave FreeBSD something to compete against the Linux KVM hypervisor. However, there were further benefits, it is small and robust as well as being performant, leaning heavily on CPU instruction sets rather than dealing with interpretation.

The initial implementation was only suitable for FreeBSD guests, and it was some time before we saw bhvye able to run other operating systems.

First up, there was Linux, and then there was a way to repackage Windows 8 or Windows Server 2012 to get them to install. This was too much for a regular user to manage and it wasn't until the arrival of the bhyve UEFI boot feature that things really took off.

UEFI boot was the killer feature that bhyve had been waiting for. This allowed for a wide range of operating systems to be installed and run on FreeBSD bhyve. When FreeBSD 11 was released, we finally had a virtualization component on par with other operating systems.

While UEFI booting was the killer feature for bhyve, the killer app for bhyve was Windows Server 2016. This was the turning point when enterprises could take bhyve and Windows in a vanilla format and have a reliable enterprise hypervisor to run business workloads in a stable fashion.

Suddenly, businesses were able to deploy equipment far and wide with a solution that was 2-clause BSD licensed and be able to tune — either via hardware or software — the hypervisor to solve their problems.

Suddenly, businesses were able to deploy equipment far and wide.

There was still a problem, however, because Windows required numerous drivers to be installed either in-image or after installation to avoid performance issues. In July 2018, this was partially solved by the implementation of the PCI-NVMe storage emulation, eventually giving bhyve the edge over KVM in storage performance for general workloads.

Today, Windows running on bhyve still requires at least the VirtIO-net drivers from RedHat to allow for network transfers to be reliable and exceed 1Gb/s. There are other drivers within the applicable Windows MSI package that is available from RedHat, and these are recommended to be loaded prior to production implementations. For Linux, most distributions have all applicable drivers, including AlmaLinux, which we are using in this article. It is possible and recommended to use NVMe emulated backed storage for Linux installations, however, it is quite difficult to configure KVM to use an emulated NVMe storage type and if you plan to move Linux workloads between bhyve and KVM, it is recommended that you set your guest to simply use VirtIO-blk storage.

The following will work for a standard FreeBSD workstation in a typical type-2 hypervisor configuration or for a dedicated FreeBSD server that is only hosting guest workloads with the applicable storage and network associated with the guests for a type-1 hypervisor.

Preparation

Typically, all modern processors from the last ten years will be suitable to use with bhyve virtualization.

Ensure that your hardware is configured with the virtualization technology enabled along with VT-d support enabled. The use of PCI pass-through is out of scope for this article, but it is recommended to enable VT-d so that it can be used when needed. After this is configured in your machine's BIOS/firmware, you can check that it visible to FreeBSD by looking for POPCNT in the Features2 of the CPU:

```
# dmesg | grep Features2
  Features2=0x7ffafbff<SSE3,PCLMULQDQ,DTES64,MON,DS_CPL,VMX,SMX,EST,TM2,SSSE3,SDBG,FMA,
  CX16,xTPR,PDCM,PCID,SSE4.1,SSE4.2,x2APIC,MOVBE,POPCNT,TSCDLT,AESNI,XSAVE,OSXSAVE,AVX,
  F16C,RDRAND>
```

Now we have confirmed that our CPU is ready, we need to install a few packages to make it easy to create and manage guest operating systems:

```
pkg install openntpd vm-bhyve bhyve-firmware
```

Briefly, OpenNTPD is a simple time daemon from the OpenBSD project. This keeps the host time from skewing. When a hypervisor is under extreme pressure from guest workloads, this can cause the regular system time to quickly get out of sync. OpenNTPD keeps time in check while ensuring your upstream time source is reporting the correct time using constraints over the HTTPS protocol. bhyve-firmware is the meta package that will load the most recently supported EDK2 Firmware for bhyve from packages. Finally, vm-bhyve is a management system for bhyve written in shell, avoiding the need for complex dependencies.

Boot strapping a machine ready for use with vm-bhyve is quite simple but attention is required for some of the ZFS options to ensure that guests remain performant on underlying storage for general workloads:

```
# zfs create -o mountpoint=/vm -o recordsize=64k zroot/vm
# cat <<EOF >> /etc/rc.conf
vm_enable="YES"
vm_dir="zfs:zroot/vm"
vm_list=""
vm_delay="30"
EOF
# vm init
```

Before we get too far into this, we should download the ISOs that we will be using later so they are ready for use by the vm-bhyve installer. To download an ISO to the vm-bhyve ISO store, use the **`vm iso`** command:

```
# vm iso https://files.bsd.engineer/Windows11-bhyve.iso
```

(sha256 - 46c6e0128d1123d5c682dfc698670e43081f6b48fcb230681512edda216d3325)

```
# vm iso https://repo.almalinux.org/almalinux/9.5/isos/x86_64/AlmaLinux-9.5-x86_64-dvd.iso
```

(sha256 - 3947accd140a2a1833b1ef2c811f8c0d48cd27624cad343992f86cfabd2474c9)

These will be downloaded into the /vm/.iso directory. Note: The AlmaLinux ISO is directly downloaded from the project and checksums can be verified upstream. The Windows11-bhyve ISO was downloaded from Microsoft and has been modified to ensure that it will install on hardware that Microsoft deems **unsupported** and has been provided to assist with this article. As such, this ISO should only be used in a lab environment. It has had the CPU and TPM requirements removed along with not needing to create a Microsoft account.

Networking

By default, vm-bhyve uses bridges to connect the systems physical interface with the tap interfaces that are assigned to each guest. When adding a physical interface to a bridge, certain features such as TCP Segment Offload (TSO) and Large Receive Offload (LRO) do not get disabled but need to be disabled for networking functions of guests to work correctly. If the host has an em(4) interface, this can be disabled by:

```
# ifconfig em0 -tso -lro
```

To avoid having to disable these after each reboot, add them to the system's /etc/rc.conf file:

```
# ifconfig_em0_ipv6="inet6 2403:5812:73e6:3::9:0 prefixlen 64 -tso -lro"
```

The above may not be required in every situation depending on the network card being used but if you experience guest network performance issues, this is what the problem will be.

To configure a vSwitch (bridge) the **switch** vm sub-command is used:

```
# vm switch create public
# vm switch add public em0
```

This creates a vSwitch called **public** and then attaches the em0 physical interface to the vSwitch.

Templates

Templates are required to assist with setting up guest configuration with the correct virtual hardware and other settings needed for them to function correctly. Using root, add the following to the templates repository:

```
# cat <<EOF > /vm/.templates/linux-uefi.conf
loader="uefi"
graphics="yes"
cpu=2
memory=1G
disk0_type="virtio-blk"
disk0_name="disk0.img"
disk0_dev="file"
graphics_listen="[::]"
graphics_res="1024x768"
```

```
xhci_mouse="yes"
utctime="yes"
virt_random="yes"
EOF

# cat <<EOF > /vm/.templates/windows-uefi.conf
loader="uefi"
graphics="yes"
cpu=2
memory=4G
disk0_type="nvme"
disk0_name="disk0.img"
disk0_dev="file"
graphics_listen="[::]"
graphics_res="1024x768"
xhci_mouse="yes"
utctime="no"
virt_random="yes"
EOF
```

Creating Guests

With storage, network, installers and templates prepared, guests can now be created.

```
# vm create -t windows-uefi -s 100G windows-guest
# vm add -d network -s public windows-guest

# vm create -t linux-uefi -s 100G linux-guest
# vm add -d network -s public linux-guest
```

The above creates both windows and linux guests with 100GB of storage allocated (using file backed storage) using their applicable templates and adds a network interface to each that is connected to the **public** vSwitch.

The windows guest needs a slight adjustment in its configuration to enable it to have network access during installation until the VirtIO drivers are installed. Edit the guests configuration and change the network interface from virtio-net to e1000:

```
# vm configure windows-guest
```

network0_type="e1000"
Revert to "virtio-net" once the RedHat VirtIO drivers have been installed.

Installing and Using Guests

To install each guest:

```
# vm install windows-guest Windows11-bhyve.iso
# vm install linux-guest AlmaLinux-9.5-x86_64-dvd.iso
```

Once the installation has been initiated, bhyve will be in a wait state. It will not commence the ISO boot process until a connection has been made to the VNC console port. To deter-

mine which guest console is running on a corresponding VNC port, use the **`list`** sub-command:

```
# vm list
NAME           DATASTORE  LOADER  CPU  MEMORY  VNC          AUTO  STATE
linux-guest    default    uefi    2    1G      [::]:5901    No    Locked (host)
windows-guest  default    uefi    2    8G      [::]:5900    No    Locked (host)
```

Using a VNC viewer like TigerVNC or TightVNC, connect the the IPv6 address and port for each of the guests to commence installation:

```
[2001:db8:1::a]:5900 # if the host is remote
[::1]:5900 # if it is on your local machine using localhost
```

After the Windows guest has been installed, the VirtIO drivers can be loaded. The drivers can be found at:
https://fedorapeople.org/groups/virt/virtio-win/direct-downloads/archive-virtio/virtio-win-0.1.266-1/virtio-win-gt-x64.msi
(sha256 – 37b9ee622cff30ad6e58dea42fd15b3acfc05fbb4158cf58d3c792b98dd61272)

Navigate to the above URL with the Edge Browser once Windows has finished installing to download and install these drivers. Once installed, shut down the host, switch the network interface in the guest configuration file back to virtio-net and the system can be started as normal.

We now have installed guests ready for use but there needs to be control over these so they can be started and stopped when required. The following commands will perform basic operations on your guests, such as starting, stopping or immediately powering off respectively:

```
# vm start linux-guest
# vm stop windows-guest
# vm poweroff windows-guest
```

The difference between stop and power off is that **`stop`** issues an ACPI shutdown request to the guest where **`poweroff`** immediately kills the bhyve process and won't shut down the guest cleanly.

Summary

This article is a brief insight into controlling bhyve and installing common operating systems with tools that are available directly from the FreeBSD package repository. vm-bhyve can do so much more than what was described here and is comprehensively detailed in the vm(8) man page.

JASON TUBNOR has over 28 years of IT industry experience in a vast range of disciplines and is currently the ICT Senior Security Lead at Latrobe Community Health Service (Victoria, Australia). Discovering Linux and Open Source in the mid 1990s, then being introduced to OpenBSD in 2000, Jason has used these tools to solve various problems in organizations that cover different industries. Jason is also a co-host on the BSDNow Podcast.