

Consolidating a Home Lab with Sylve

A browser-based management plane for bhyve, jails, and ZFS on FreeBSD

Sven Rüdiger

I'd been looking for an “easy way” to run virtual machines natively on my FreeBSD servers for years. The pieces were all there: bhyve is a good hypervisor, ZFS does what ZFS does, and nobody needs to be sold on the FreeBSD network stack. But actually using them together, day to day, still meant shell scripts and config files I maintained by hand. There was no interface that sat on top of the system I already had without trying to take it over.

I watched the [talk about Sylve](#) at [bhyveCon 2025](#) and tried it the same day I heard the talk. That evening I had Windows running as a bhyve guest on one of my dev servers, and I wrote [a short blog post](#) about it mostly because I couldn't quite believe how little had gone wrong.

This article is about how I used Sylve to tidy up a home lab that had gotten out of hand, why it suits the way I already run FreeBSD, and where things landed after I moved my last standalone box onto it. There's a step-by-step install section at the end.

The problem I was trying to solve

My lab grew the way most home labs grow (just a little faster), which is to say without much of a plan. For years I ran Home Assistant straight on FreeBSD. It was fine until it wasn't. About two years ago the Python dependencies turned into a porting battle at every upgrade, and keeping it healthy started eating more evenings than it was worth. Home Assistant isn't unusual in that respect; plenty of projects only really expect a Linux host, and I'd quietly collected a handful of workloads in the same boat.

Virtualization was the obvious fix. I'd poked at bhyve before, and I'll be honest that the problems were mostly mine and not the hypervisor's. Managing them by hand was a pain and vm-bhyve helped just a little. Instead of forcing it, I went sideways: I put the Linux workloads on Raspberry Pi CM4 boards, one job per board. That worked, but now I had a little shelf of single-purpose machines, called ComputeBlades, . Then the CM5 support never showed up, so the future of this approach was not clear.

What I actually wanted was to pull those Linux workloads back onto FreeBSD hosts I understood, along with the one or two proprietary Windows tools on an Intel NUC I can't get away from, and to do it without throwing away everything I'd learned over the years. Sylve is what closed that gap.

Why Sylve fits a FreeBSD-first setup

The thing I like most about Sylve is that it doesn't try to be the platform. Everything it drives already lives in FreeBSD's base system. bhyve is the hypervisor, ZFS is the storage, and the network stack handles the plumbing. Sylve just manages those pieces, orchestrating VMs, jails, datasets, virtual switches, and DHCP through a Go backend and a REST API. The OS underneath stays untouched.

That's really the whole case against reaching for something like Proxmox. Proxmox is a fine product, but running it would have meant giving up FreeBSD, and with it all the habits and muscle memory I lean on every day.

Sylve lets me keep my hosts. The features I care about stay in base, where they're well understood. Sylve is just the manager sitting on top, much like Bastille is for jails, and that's all I want it to be. I don't need a whole hypervisor distribution just to get a UI, and ZFS snapshots and replication are still plain ZFS instead of something buried under a vendor's abstraction.

First impressions and early development

I picked Sylve up very early, but never regretted it. There were UI bugs and the usual rough edges of software, which is exactly what you'd expect from a young, experimental project. What surprised me was that the problems didn't touch the VMs. Even when I screwed up my Sylve installation, the workloads underneath just kept running. Because Sylve manages base-system pieces instead of replacing them, a broken UI never turned into a dead VM. That gap between the management layer and the actual workloads is what convinced me to keep using it for real rather than as a toy. It's come a long way since then. When I started I was building from the GitHub source; these days it's packaged and the install is a lot shorter.

Consolidation in practice

Once Sylve was in place the cleanup went fast. Spinning up a few Linux guests took no time, and I added a Windows guest to run a piece of proprietary security software that has no FreeBSD version. The workflow that never stuck for me with hand-rolled bhyve suddenly worked, because now it was just a form: CPU, memory, a datastore, a switch, and a console to watch the installer. A couple of weeks ago I moved my last standalone box into Sylve, and with that the shelf of single-purpose machines basically folded down into a few capable FreeBSD hosts.

One thing is worth calling out, because it's a good sign of how Sylve behaves on a system that already has a history. It comes with its own jail manager, but I'd been running a couple dozen jails with Bastille for a long time. I didn't want to migrate them, and it turned out I didn't have to. The two coexist without any fuss. I keep my existing jails in Bastille and let Sylve handle the

VMs. Leaving the old jails where they already worked meant not spending extra time for migration. That would have bought me nothing, and it kept the setup about as simple as it can be. With that comes my usual approach: fewer dependencies, and let each tool do the part it's actually good at.

Maintenance follows from the same idea. With pkgbase underneath and boot environments to fall back on, keeping the platform current is easy. I can stage an upgrade in a fresh boot environment and roll it back if it misbehaves, and since Sylve doesn't own the base system, updating the host doesn't turn into a fight with the management layer.

Results

The short version: I went from a scatter of single-purpose boxes, Pi boards and NUCs, down to a couple of FreeBSD servers. The Linux workloads that pushed me toward the CM4 detour are back on hardware and an OS I know best, including the Windows VMs. My Bastille jails stayed untouched.

One more thing worth being upfront about: Sylve can also run in a cluster as well. By the time of writing the feature for migrating VMs between hosts was just released and it works with little limitations, like the requirement for same ZFS pool names on both hosts. For my home lab, the 1-click migration is a great support already.

If you're a FreeBSD user who's been holding out for a bhyve UI that works with your system, Sylve is probably the tool you have been looking for.

Installing Sylve

System requirements

Sylve just needs **FreeBSD 15.0 or later** and a **ZFS pool** to work with. One CPU and 512 MB of RAM should carry a jails-only host, but if you want to run bhyve VMs, count on at least **2 vCPUs and 4 GB of RAM** and a ZFS pool of 20 GB or more. Worth knowing: a lot of VPS instances don't do nested virtualization, so

bhyve VMs may be off the table there. Why should you nest a hypervisor in a VM anyway? Jails run anywhere.

Installing the package

Sylve is packaged now, so the install itself is one command on FreeBSD 15 or newer:

```
pkg install sylve
```

Out of the box, Sylve comes up over HTTPS using its own built-in certificates, so you can reach the UI straight after install without generating anything. It can still serve plain HTTP as well, but I leave that out of my config and keep it HTTPS-only. If you'd rather not run the bundled certs and if you don't run your own CA, creating self-signed certificates can be done via Sylve UI.

It's self-signed, so your browser will complain the first time; accept the exception or import the cert into your trust store to quiet it down. For a host that's reachable from outside, just use the built-in Let's Encrypt or any public CA, but keep in mind that you might not want to put your Hypervisor's management online, especially not for a project as young as this one.

Creating the configuration

Sylve reads a JSON config file. Edit the default `config.json` in `/usr/local/etc/sylve` with the basics, plus the paths to the cert and key if you're supplying your own:

```
{
  "environment": "production",
  "proxyToVite": false,
  "auth": {
    "enablePAM": true
  },
  "admin": {
    "email": "admin@sylve.local",
```

```
"password": "your-strong-password"
},
"logLevel": 0,
"port": 8181,
"raft": {
  "reset": false
}
}
```

A couple of notes on this. There's no `httpPort` in here on purpose: Sylve will serve plain HTTP if you add one, but I keep it HTTPS-only and leave it out. `enablePAM` is `true`, which lets you log in with system accounts — set it `false` if only the admin user (and whoever you add later in the UI) should get in. `logLevel 0` logs everything while you're setting up; `3` once things settle, keeps it to errors. A Cluster needs at least three nodes. By setting `raft reset` to `true` you can easily rebuild your cluster, without losing the VMs.

Loader and kernel tunables

If jails and ZFS are in the plan, add these to `/boot/loader.conf` so ZFS loads and resource accounting is on:

```
zfs_load="YES"
kern.racct.enable=1
```

I'd also cap the ZFS ARC so it doesn't starve your guests. On a 4 GB box, 512 MB is a reasonable starting point but should be tuned to your workloads via Sylve UI. Too small and ZFS slows down; too big and it eats RAM.

Enabling and starting Sylve

The package already dropped the `rc.d` script in place, so all that's left is to enable the service and start it:

```
sysrc sylve_enable=YES
service sylve start
```

I'd actually reboot here rather than just starting it, both to check the loader tunables took and to confirm Sylve comes up on its own. Either way, it should then answer at <https://<your-server-ip>:8181>. Log in with the `admin` credentials from `config.json`.

Creating a virtual switch

Before you can create your first VM, you need a virtual switch for it to attach to. In the UI, go to Network → Switches → Standard → + New, and for most home setups enable DHCP on it. Any VMs and jails you make later hang their networking off this switch.

Your first VM: Windows

Here's the short version of running Windows on Sylve, pulled from the writeup I did on my blog.

Start with storage: create a ZFS volume for the VM under Datacenter → your host → Storage → ZFS → Datasets → Volumes → + New, sized however you like. Then grab a Windows 10 ISO (the LTSC evaluation build is handy) and pull it in through Utilities → Downloader → + New by pasting the download link.

Now Create VM. On the Storage tab, pick ZFS and the volume you just made, and choose NVMe as the disk bus for Windows 10 (Windows 11 is happier on AHCI for the moment). On the Network tab, select your switch and VirtIO emulation. On the Advanced tab, copy the VNC password if you want to reach the console from an external VNC client. Create it, open Summary, and hit Start.

To reach the installer you either catch the boot prompt in time, or drop into the UEFI shell and launch it yourself:

```
fs0:
cd EFI\BOOT          # or EFI\Microsoft\Boot, depending on the ISO
BOOTX64.EFI          # or bootmgfw.efi
```

Once Windows is up, install the VirtIO drivers. Download them the same way through the Downloader, mount the ISO inside Windows, and run the installer. If Windows can't see the disk or the NIC, that's nearly always the missing VirtIO drivers, so add them during or after setup and it sorts itself out.

Conclusion

Sylve helped me to shrink the number of FreeBSD nodes running in my home lab from 16 down to four. Everything now runs on a couple of FreeBSD hosts: the Linux workloads, the Windows VM, and my existing Bastille jails, all side by side without stepping on each other.

The one downside is power — Intel hardware idles higher than a stack of ARM boards did. But as node count grows, that gap narrows. Out-of-band management on the hosts is a nice bonus for maintaining the node's firmware and in cases of failures (that might even happen with boot environments).

What's next is mostly watching Sylve's clustering, especially live migration, mature. By the time of finishing this article, 0.3.0 just landed with basic migration features and even better documentation <https://sylve.io/>.

Sven is a longtime FreeBSD user since 2000, with a deep-rooted passion for open-source technology. Over the past decade, he has actively contributed to the FreeBSD Ports collection, and for the last five years has served as a port maintainer. Holding an M.Sc. in Data Science, Sven enjoys bringing big data and data science tools into the FreeBSD ecosystem.

Copyright © 2026 Held by Owner/Author. Publication Rights Licensed to The FreeBSD Journal. This work is licensed under a [**Creative Commons Attribution International 4.0 License**](#).

3980 Broadway Street
STE #103-107
Boulder CO 80304
1-720-207-5142
info@freebsd.foundation.org

