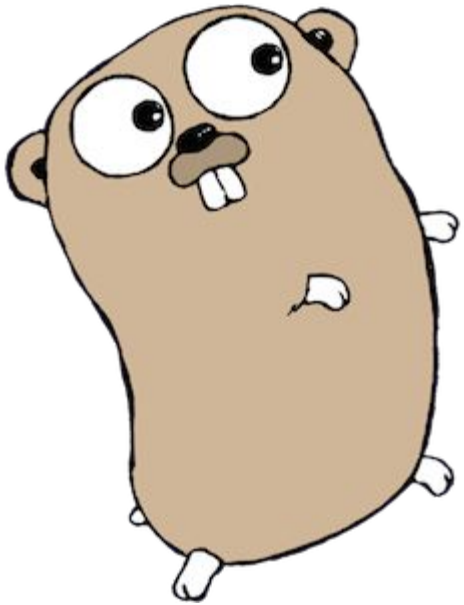


My 2025 Linux + Go self-hosting stack

Michael Stapelberg
https://michael.stapelberg.ch
https://mas.to/@zekjur

GPN, 2025-June-21

Agenda

- Part 1: What is self-hosting? Why do I self-host?
- Part 2: What services do I (not) self-host? What do I self-build?
- Part 3: Home Automation / Smart Home
- Part 4: gokrazy appliance platform
- Part 5: Web hosting (public and private)

Part 1: What is self-hosting?

What is self-hosting?

- Running services yourself, instead of relying on others.
- This is not a new concept: I ran my first self-hosted web server out of my child bedroom on a PC with the Apache web server ca. 2001 — almost 25 years ago!
- In this talk, I want to explain why I self-host, what services I self-host / self-build (or not!) and how I go about that. Spoiler Alert: It’s all Linux, and mostly Go!

Context: Why do I self-host?

- More **privacy**: nobody else can access your private data
- More **control**: providers can suddenly (have to) stop service,
  but if I run my services myself, I am in control (“business continuity”)
- More **speed**: self-hosting is often more local (same LAN, or network-close),
  and provisioned with more capacity per user
- More **stability** (not reliability): things change on your schedule
  …but of course you’re still using software that someone else is developing

Part 2: What services do I self-host?

What do I self-host: Media Library 🎬

- Because physical media will become unreadable one day,
  I had always backed up my movies and TV series to disk, since the 2000s

- Back then: Kodi, a media browser/player program, on a “HTPC” (Home Theater PC)
  connected via HDMI to a TV, with an infrared remote,
  mounting my NAS (Network Attached Storage) via Samba (SMB/CIFS)

- Nowadays: Jellyfin tvOS or Android TV app streams media
  from the Jellyfin server running on my NAS (Network Attached Storage)

- Benefit: Your own catalog will never remove your favorite movies and series!
  Disadvantage: collection needs to be extended and consumes disk space

What do I self-host: Music streaming 🎵

- Used Google Play Music because one could upload own media (**and** stream!)
  Then the service was shut down in favor of YouTube Music — without upload feature :(

- Back then: Paid subscription to Plex, Plex:Amp on iPhone,
  running the Plex Docker container on my NAS
  👎 annoying bugs (e.g. in gapless playback), app never got better

- Nowadays: Navidrome on my server, amperfy on iPhone
  👍 amperfy is really nice! Navidrome web interface works OK

- Benefit/Disadvantage: Higher cost to buy music, but the streaming media business model seems broken — more profit for the artist is a good thing!

What do I self-host: Photo library 📷

- Used Google Photos, which had a great UI and surprisingly good search

- Nowadays: Immich on my server, Immich App on iPhone
  👍 similarly good UI, and much faster!

- Benefit: aside from privacy and speed,
  With Open Source software, you will always have the freedom to remove any limitations you don't like — for example, Cannabis was legalized in Germany, and yet Google Photos shows no search results for "Cannabis" (when Immich does!)

What do I self-host: Static Web Sites 🌐

- For Open Source Projects, I used to host static files via GitHub Pages,
  so that contributors will be familiar with the infrastructure and can modify it
  - I used to use CloudFlare for DNS hosting and Content Distribution / Caching
  - Attractive offering: Free, can add TLS, rewrite rules, an API for DNS updates
  - 👎 dependencies on US services (currently?) pose a continuity risk

- Nowadays: Caddy Web Server and CoreDNS Server running on my router

- Benefit: easy to configure and much faster than the previous setup, at least in Europe

What do I self-build: Smart Home Automation 🏠

- Intentionally picked (somewhat) open systems so that I can interface with them.
- MQTT as central message bus, “regelwerk” integrates components:
  shelly2mqtt, hue2mqtt, mystrom2mqtt, nuki2mqtt
- Benefit/Cost: Full control over functionality / behavior, but more effort to build than to configure an off-the-shelf solution like Home Assistant

What do I self-build: Daily Backups ⤴️

- I am a backup minimalist: using rsync instead of Borg Backup, rclone, Bacula, …
  - Want per-file access without custom tools, weird FUSE file systems or similar
  - Want zero setup needed to restore data, any live Linux will do
  - Want to avoid circular dependencies (e.g. rclone keyfile only inside backup)
  - Using rsync's `--link-dest` flag for incremental backups

- Storage: Linux + LUKS + [Ext4 | ZFS]
  - The slowest operation is deleting old backups (fast on SSDs!)

- Following 3-2-1 rule: 2 copies on separate media (two NAS builds), one offsite

Part 2: What services do I **not** self-host?

What do I **not** self-host: E-Mail 📧

- I use my own domains (e.g. michael@i3wm.org), but outsource the e-mail hosting

- For a while, I ran my own mail server, but then it died while I was hundreds of kilometers away without remote hands and urgently needed to coordinate the details of my upcoming out-of-country internship…
  - I need my email to be more reliable than that
  - Running email reliably takes a lot of time and effort!

- The Gmail Spam filter was (is?) so much better than Open Source solutions back then

What do I **not** self-host: Mastodon 

- Running a one-person instance burdens one with federation policy and abuse handling

- I can move my account to another server and keep my followers.
  I don’t care much about my posts, the important stuff is recorded in my blog.

Part 3: Home Automation / Smart Home

Personal context

- full-time work leaves little time for other things
- …but I have a little more time than others who have kids!
  …and I find this stuff fascinating; you might have other hobbies
- I prefer simple solutions that stand the test of time
  over complex solutions that are flashy but unreliable

Goals

- hands-off as much as possible
  - self-documenting as much as possible
  - as uniform as possible

- quick and painless recovery
  - after a power outage, everything must come up! (automatically, eventually)
  - → power cycling must always be a viable troubleshooting strategy

- failure domains! must be able to work on small systems in small increments of time
  - privacy domains! how sensitive is which kind of data? count on a breach

Home Automation

- Tried out a bunch of systems over the years: https://michael.stapelberg.ch/posts/2022-03-19-smart-home-components/

- Preference for developer-friendly devices (e.g. Shelly Energy Meter supports MQTT) …or at least have an official API (e.g. Philips Hue) with a significant user base
  - Using Zigbee directly is too inconvenient (sparse docs, weird issues)

- Strategy: Centralize everything on MQTT as the single message bus
  - MQTT is easy to integrate into your own microcontroller projects
  - Some vendors support MQTT by default! 🥳
  - For others: use zigbee2mqtt or custom-built (I wrote my own hue2mqtt, nuki2mqtt, etc.)

Home Automation (2)

- Off-the-shelf solutions are Home Assistant or Node-RED,
  but I enjoy writing/maintaining Go code more than yet another DSL / UI

- regelwerk has a bunch of control loops that all fulfill this interface:

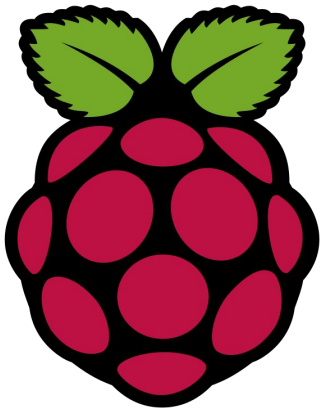
```
type controlLoop interface {
    StatusString() string // for human introspection
    ProcessEvent(MQTTEvent) []MQTTPublish
}
```

- → https://michael.stapelberg.ch/posts/2021-01-10-mqtt-introduction/

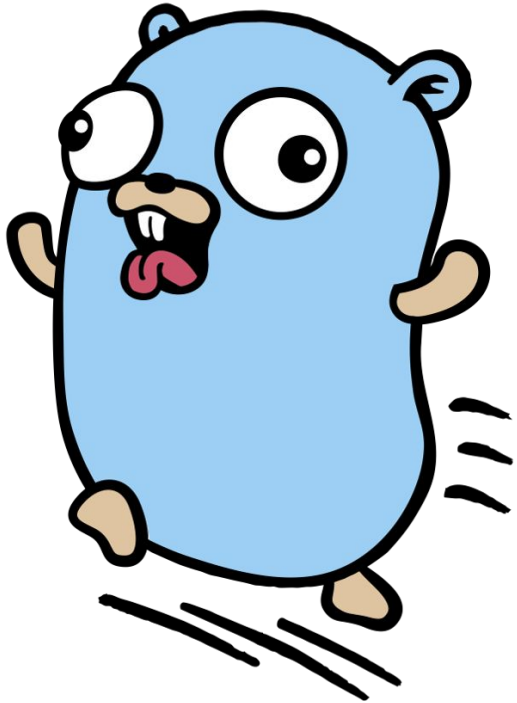
Home Automation (3)

- Scenario: Shelly Motion Sensor turns on Philips Hue lights. What do we need?
  - shelly2mqtt accepting HTTP requests from Shelly Motion Sensor
  - regelwerk detects motion, sends MQTT command to turn on Philips Hue
  - hue2mqtt turns MQTT message into Philips Hue API request
  - → 3 Go programs that need to run 24/7

- Best choice: Raspberry Pi or other SBC
  - Running Go is easy, even on the Pi 1 from 2012
  - Smaller devices require TinyGo or similar
    (I'd use that to build a keyboard!)



Part 4: gokrazy appliance platform



gokrazy: origin story

- Back in 2012, the release of the Raspberry Pi was transformative for hobby projects: bye-bye microcontrollers, hello Linux with Go 🥰

- For a “smart lock” project at RaumZeitLabor, we needed a Pi appliance
  → wrote a shell script that builds a read-only Debian installation
  → worked reliably, but was effectively too time-consuming to update

- In 2017, I experimented with a better approach, specifically for Go

gokrazy: origin story

- My introduction to Operating System development (osdev) was when I realized `CGO_ENABLED=0 GOARCH=arm64 go build hello.go` results in a binary that the Pi kernel will run (with `init=/hello`)! 🤯
  - Second key factor: Linux upstream kernel supports the Pi 3 natively

- Could we have a system that can easily be updated over the network, is stateless/read-only by default and deploys Go packages?

gokrazy

- from-scratch appliance platform built entirely in Go

  Linux kernel + (firmware) + gokrazy Go userland + <your app(s)>

  no C userland or runtime environment! no glibc, OpenSSL, xz, …

- `gok new` # create ~/gokrazy/hello/config.json

  `gok overwrite --full /dev/sdx` # write SD card for Raspberry Pi

- → gokrazy.org/quickstart

gokrazy

built at 2025-06-08T17:38:27+02:00

host	dr
kernel	Linux 6.15.1 (aarch64)
model	Raspberry Pi 4 Model B Rev 1.1
SBOM	a467a9ccb9
EEPROM	68c96926f7
(SHA256)	548581c70a

gokrazy revisions, hostname / model, kernel version

services

path	status	last log line
/gokrazy/dhcp	running	stderr: 1970/01/01 01:00:09 dhcp.go:255: adjusting route [dst=10.0.0.0/24 src=…
/gokrazy/ntp	running	stderr: 2025/06/08 17:39:13 ntp.go:33: clock set to 2025-06-08 15:39:13.99472…
/gokrazy/randomd	running	stderr: 1970/01/01 01:00:03 randomd.go:38: seeding RNG from /perm/random.…
/user/hmgo	running	stderr: 2025/06/08 17:39:39 entering BidCoS packet handling main loop
/user/hmq	running	stderr: {"level":"info","timestamp":"2025-06-08T17:39:38.578+0200","logger":"br…
/user/hue2mqtt	running	stderr: 2025/06/08 17:39:38 http.ListenAndServe(":8779")
/user/mystrom2mqtt	running	stderr: 2025/06/08 17:41:08 published to MQTT
/user/nuki2mqtt	running	stderr: 2025/06/08 17:39:38 http.ListenAndServe(":8319")
/user/regelwerk	running	stderr: 2025/06/08 17:41:17 regelwerk.go:170: MQTT: shellies/shellyem3-C45B…
/user/shelly2mqtt	running	stderr: 2025/06/08 17:39:38 Subscribing to github.com/stapelberg/shelly2mqtt/c…

gokrazy minimal
100% Go userland

home automation
Go services 😊

memory

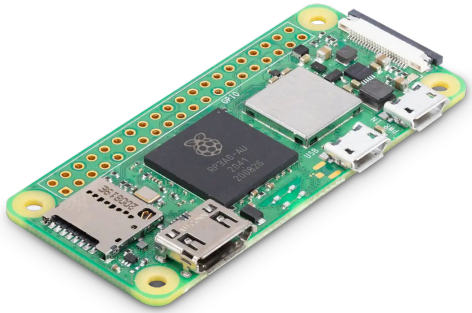
1821.9 MiB total, 1319.2 MiB available

resident set size (RSS) by service:

storage

/dev/mmcblk1p4: 28.0 GiB total, 0.3 GiB used, 26.3 GiB avail

gokrazy: the supported SBC landscape



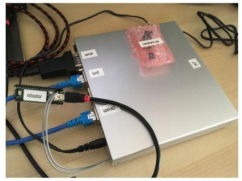
- The cheapest and smallest supported platform is the Raspberry Pi Zero 2 W, which runs at ≈1W!
  You can power it via (micro) USB and connect it via WiFi
  (gokrazy supports encrypted WiFi without `wpa_supplicant` or similar)

- Personally, I prefer ethernet over WiFi whenever possible,
  so I prefer running my automation on the Raspberry Pi 3 or Pi 4
  The Pi 5 is nice, but not yet supported by upstream Linux 😞

All currently supported platforms

Raspberry Pi 5

Raspberry Pi 4B



PC Engines APU

Raspberry Pi Zero 2W

Raspberry Pi 3 B and 3 B+

gokrazy kernel repository map

Officially supported kernels

These kernels are tested in Continuous Integration.

They are built from one of two sources: Either from Raspberry Pi's Linux kernel fork, or from the upstream Linux kernel from kernel.org.

The upstream Linux kernel gets security fixes the quickest, but the Raspberry Pi Linux kernel supports more Raspberry Pi models (notably the Pi 5, which upstream does not support) and peripherals.

repository	source	devices
gokrazy/kernel.rpi	Raspberry Pi	Pi 3, Pi 4, Pi 5, Pi Zero 2 W
gokrazy/kernel	kernel.org	Pi 3, Pi 4, Pi Zero 2 W
gokrazy/kernel.amd64	kernel.org	PC x86_64, VMs
gokrazy/kernel.arm64	kernel.org	PC arm64, VMs

Community-supported kernels

Independently from the officially supported gokrazy kernels listed above, people of our community provide alternative kernels and firmwares, in order to run gokrazy on unsupported platform or to provide new features. They may not be as thoroughly tested as the official platforms. Please report any issue to their respective repository.

repository	source	devices
gokrazy-community/kernel-rpi-os-32	Raspberry Pi	Pi 1, Pi 2, Pi 3, Pi 4, Pi 5, Pi Zero 2 W (32 bit)
anupcshan/gokrazy-odroidxu4-kernel	kernel.org	Odroid XU4, HC1, HC2
anupcshan/gokrazy-rock64-kernel	kernel.org	Pine64 Rock64

I recommend using the upstream kernel from kernel.org for quickest security fixes!

gokrazy: supply chain

- github.com/gokrazy/kernel — auto-updated Linux upstream kernel
  github.com/gokrazy/firmware — auto-updated Raspberry Pi firmware files

- `gok get` wraps `go get`, all gokrazy programs are Go modules
  can use `govulncheck` for security analysis, `replace` for patching, etc.

- `gok update` updates an instance over HTTP(S)
  automate it by configuring a systemd timer, cron job, or similar

gokrazy: system properties and advantages

- declarative configuration (`config.json`),
  builds are fully reproducible and “version-locked” (thanks to Go!),
  state is very clearly separated (in `/perm` partition)

- minimal time to recovery: `gok overwrite` and restore `/perm` backup
  I keep a spare SD card, so when one dies, I need <1min to deploy a new one

Part 5: Web Hosting (public services)

What do we need for web hosting?

- An internet connection!
  Did you notice that my smart home does not need the internet? :)

- Sufficiently-powered hardware
  Maybe more than a Pi?
  More reliable storage than SD cards (or USB sticks) would be nice!

- Server software, and a way to run it (gokrazy?)

Internet connection

- For hosting private services (for yourself), any internet connection is fine!
  Even your home internet connection will probably be fast enough!

- For hosting public services, I would recommend FTTH for low latency
  Even better if your connection is unmetered (= flatrate)

- Neither are deal breakers, just inconvenient:
  Limited speed? Set up traffic shaping / QoS
  Limited traffic? Set up rate limiting / monitoring

Internet connection: IP addresses

- Does your ISP give you a public IPv4 and/or IPv6 address?
  - Maybe you only need IPv4 (or IPv6!), or tunnel the other one
  - Maybe your ISP uses CGNAT, so you don’t have a public IP address at all
  - → Call the support line and tell them your online gaming doesn’t work :)

- Does your IP address change?
  - Set up DynDNS to get a name to point to your current IP address

- Are incoming connections allowed? Technical or contractual level

- If your home internet connection is not good enough, go where the internet is (community rack, providers)

Hardware selection / considerations

- Power usage: how many Watts does it draw out of the wall?
- Space usage: how much physical space does the device need?
- Low Noise: is the device whisper-quiet?
  I want to be able to sleep in the same room.
- Upgrade-/Repairability: can we add extension cards? More RAM? More disk?
- Performance: how fast is this machine?

Hardware: Spectrum of Compute Power

- **microcontroller** like ESP32: low single-digit watts
  single-board computers (**SBC**) like the Raspberry Pi: mid single-digit watts
  [~] performance [-] upgradability [+] peripherals [+] low-noise [~] space-efficient
  [-] needs a case [+] low power

- **Mini PCs**: low double-digit watts
  [+] performance [+] upgradability [+] low-noise [+] space-efficient [+] low power
  Full-size PCs: high double-digit watts
  [++] performance [++] upgradability [+] low-noise [-] space-inefficient [-] high power

- **(19") Servers**: triple-digit watts
  [~] efficiency over performance [-] noisy [+] space-efficient [-] high power

Mini PCs are great!

- → blog post: Ryzen 7 Mini-PC makes a power-efficient VM host (2024)
- These things are fast! Much faster than the VM I rented! 8 cores, 64G RAM, 2x M.2 SSD storage
- If a Raspberry Pi doesn't cut it, this one is likely the sweet spot for most people — fits into a shelf.

My 25 Gbit/s router / server

- init7 introduced 25 Gbit/s FTTH,
  so of course I had to upgrade (2022)
- → blog post: 25 Gigabit Linux
  internet router PC build (2021)
- needed more PCIe slots than
  Mini-PCs offered for fast NICs,
  also combined router + server
- Linux can do 25G without tuning!
  (be sure your system uses TCP BBR)

Software: router7

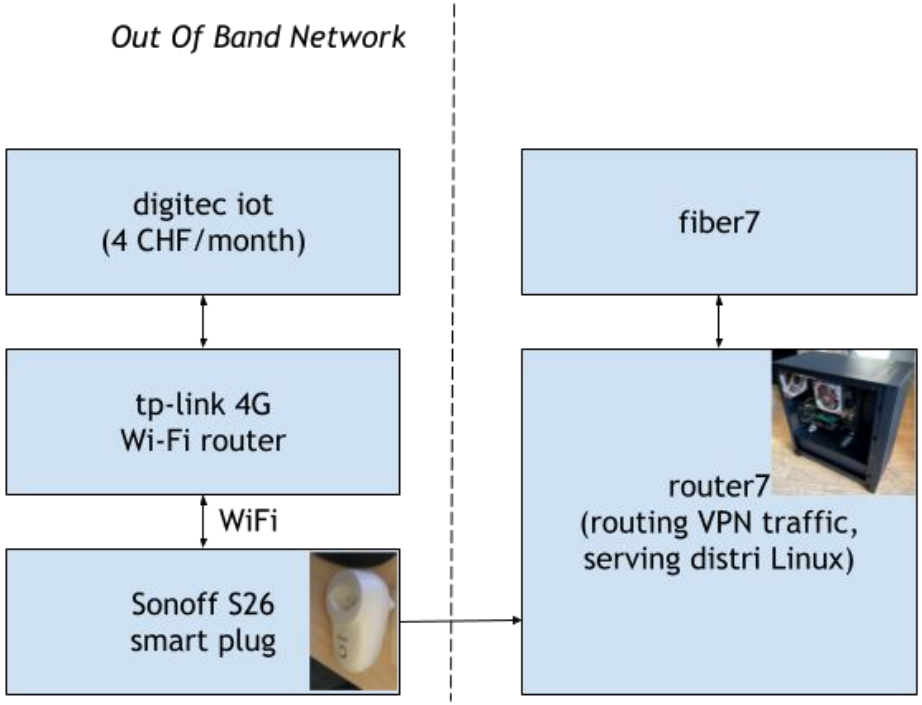
- Used to use the Turris Omnia router (Open Source / Open Hardware)

- One of their automated updates breaks my IPv6 connectivity 😢
  → neither the odhcp6c author nor init7 wanted to fix it

- Ultimately it was faster to implement my own router than to wait for a fix!
  - prototype: connect another USB-to-ethernet to a Pi running gokrazy
  - enable IP forwarding (and NAT for IPv4)
  - …and implement the control plane (DHCPv4 client, DHCPv6 client, DNS forwarder, DHCPv4 server, IPv6-RA server)

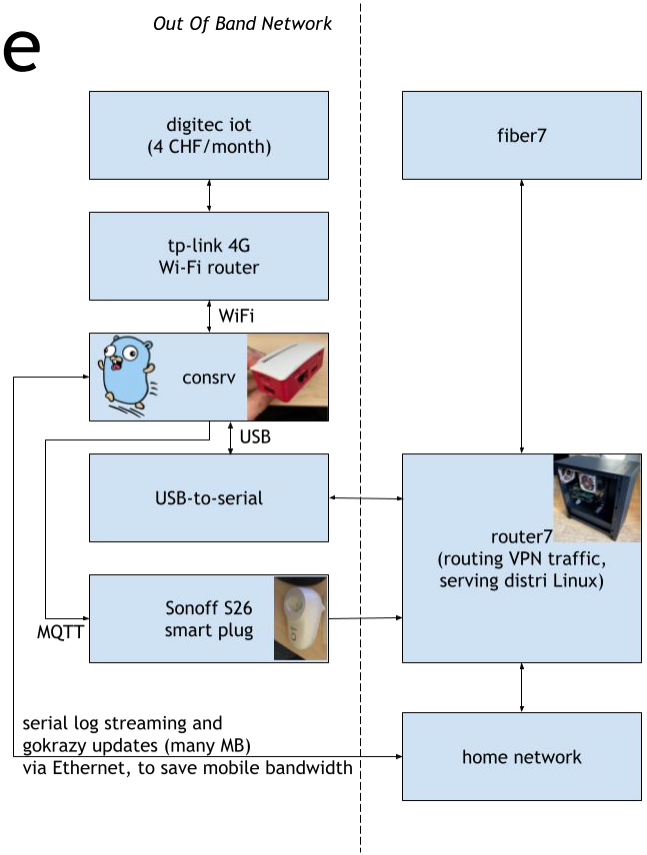
out-of-band access: power

- → blog post: DIY out-of-band management: remote console server (2022)
- Today's IOT devices like Smart Plugs can make for very pragmatic remote management solutions :)
- Even better than power on/off: remote serial console (usb2serial)



out-of-band access: serial console

- The Pi Zero 2W runs at <1W
  → 24/7 serial console streaming
- Matt Layher's consrv is an SSH to Serial Console bridge, allowing convenient remote access
- Tailscale mesh VPN for connectivity through CGNAT on mobile uplinks and access control



A step up: TinyPilot, PiKVM, etc.

- If you want HDMI access, remote keyboard access and remote media emulation (ISO images as CD):
  Use something like a TinyPilot

- **But:** more expensive in comparison!
  TinyPilot at 400 USD, PiKVM at >200

- Smart Plug, Mobile Router, Pi 0 2W comes out at a little over 100

web hosting: Caddy

- Caddy web server makes configuration and HTTPS certificate management so easy!

- This is the (full!) Caddyfile I'm using to serve i3wm.org:
  (I'm running `caddy run --config /etc/caddy/Caddyfile`):

```
https://i3wm.org {
    log {
        output file /perm/logs/access-i3wm.org.log
        format transform "{common_log}"
    }
    root * /perm/srv/i3wm.org/docs/
    file_server
}
```

web hosting: updating

- For some projects, my router uses git to mirror a repository.

- For others, I have a push script that combines hugo+rsync:
  (medium-term, adding native hugo support to my mirror program would be nice)

```
#!/bin/sh
hugo
rsync \
  --exclude .git \
  -av \
  ./docs/ \
  router7:/perm/srv/michael.stapelberg.ch/
```

DNS hosting: CoreDNS

- CoreDNS is a Caddy-inspired DNS server :)
- Very easy to serve a (managed) directory of zone files! 😍

```
.:53 {
  bind 2001:db8::53
  auto {
    directory /perm/srv/stapeldns/output-generated
    reload 10s
  }
  errors
  log
}
```

DNS hosting: DNS zone writer

- Small loop around Go `text/template`: provides public IPv4 address to template and increments the zone SOA serial on changes, which triggers a CoreDNS update
  Example: `tmpl/tmpl.i3wm.org`:

```
$TTL 3600
@   3600    IN  SOA ns.inwx.de. hostmaster.inwx.de. {{
.Serial }} 10000 2400 604800 3600
@   86400   IN  NS  ns.inwx.de.
@   86400   IN  NS  ns2.inwx.de.
@   300 IN  A   {{ .Public4 }}
@   IN  AAAA    2a02:168:4a00::d1d1
www 300 IN  A   {{ .Public4 }}
www IN  AAAA    2a02:168:4a00::d1d1
```

DNS hosting: INWX hidden primary

```
.:53 {
  bind 2001:db8::53
  auto {
    directory /perm/srv/stapeldns/output-generated
    reload 10s
  }
  acl {
    allow net 2a0a:c980::53
    block
  }
  transfer {
    to 2a0a:c980::53  # send NOTIFY requests
  }
  // …
```

DNS hosting: IPv4 DynDNS for free

- init7 provides a static IPv6 subnet upon request,
  and INWX supports IPv4 or IPv6 for the hidden primary

- When my IPv4 address changes, I use zone transfers
  to push changes to my “DynDNS provider” ;)

Part 5: Web Hosting (private services)

Old approach: commercial NAS

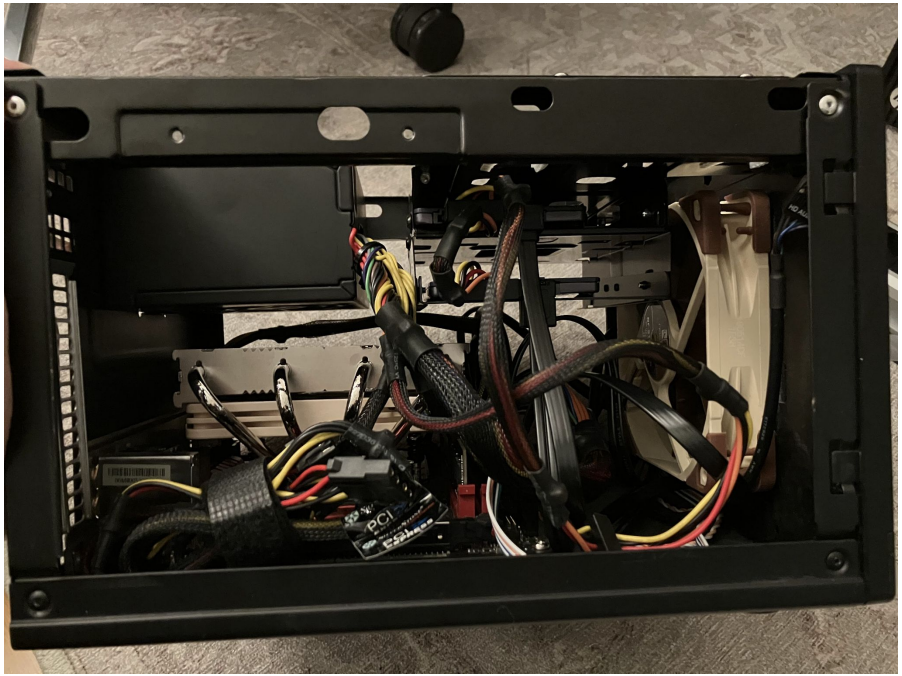
- In 2013, I tried using an off-the-shelf commercial NAS
  The QNAP was comparatively well-explored and FOSS-compatible: blog posts about how to net-install Debian, and instructions for how to solder a serial port.

- While *technically possible*, all these speed-bumps resulted in not feeling well-equipped for maintenance/recovery and hence not updating very often 😕

- There's *inadvertently closed* and *intentionally closed*:
  Synology no longer allows you to use any disk you want

New approach: self-built PC NAS

- → blog post: My 2023 all-flash ZFS NAS (Network Storage) build
- Very similar to the Ryzen Mini-PC, but with more space for SATA SSDs
- HDMI and USB connectivity makes for super-easy maintenance
- Samba is easy to set up (file share), rsync (backups), Jellyfin (media)



NixOS

- I used to use CoreOS, then Flatcar (declarative)
  But: Flatcar does not support docker-compose,
  so I could not run Immich 😕
  → started looking for an alternative

- Wanted something that mostly works like gokrazy,
  but gives me access to more non-Go software

- → blog post: How I like to install NixOS (declaratively) (June 2025)

NixOS: Trying immich

- The level of abstraction in NixOS is higher than elsewhere, and the abstractions seem to work better (compose better)

- Trying out Immich is extremely simple. configuration.nix diff:

```
+  services.immich = {
+    enable = true;
+    host = "immich.example.ts.net";
+  };
```

Nix/NixOS: Trying immich

- Nix itself is nicely self-contained and portable, so on my Arch Linux PC,
  I can use the same Immich package from nixpkgs to upload my photos:

```
% nix run nixpkgs#immich-cli -- \
    login https://immich.example.ts.net secret-apikey

% nix run nixpkgs#immich-cli -- \
    upload --recursive /home/michael/lib/photo
[…]
```

Conclusion

- I can recommend:
  declarative configuration
  the appliance model (read-only root file system)
  clearly separating state (/perm partition)
  running daily backups (3-2-1 rule)
  writing services in Go / using Go software

- I caution against:
  vendor lock-in or too-specialized setups / platforms
  mixing things carelessly between failure / privacy domains

The End; now self-host/self-build something! 🚀

- I hope I could inspire you to run something (anything!) on a small(-ish) computer!
- Questions? Talk to me after the presentation :)
- Give me feedback on this presentation!