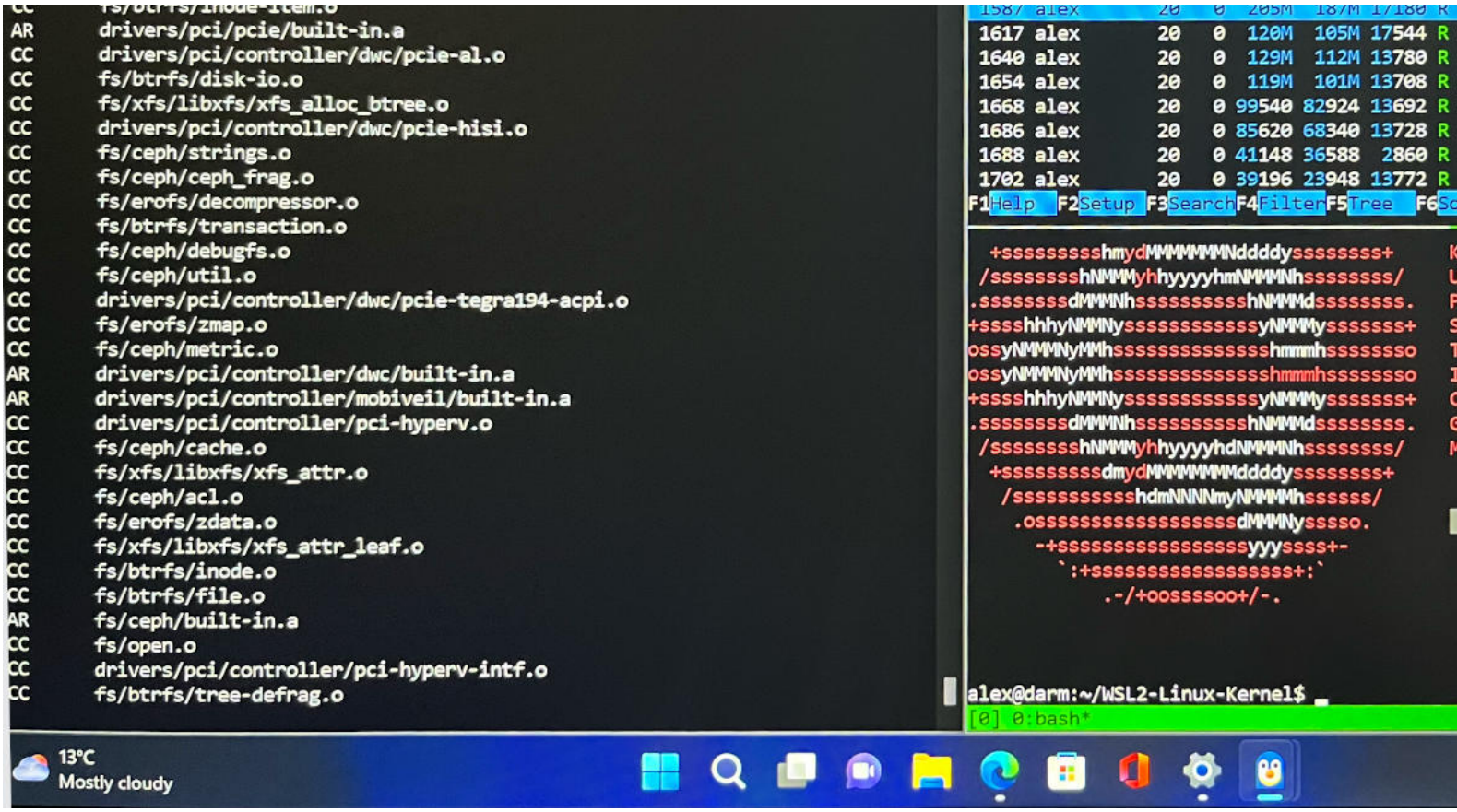
Linux on Microsoft Dev Kit 2023

This article was fetched from an rss feed(31/10/2022)



When I heard about the Microsoft Dev Kit 2023, I was surprised by how generous the specifications were for the price? Naturally, I wanted to know if I could run Linux on it. You may also wonder. The answer is: kinda.

I'm not sure why you are interested in ARM computers, but for me I got involved with them when porting software to the Raspberry Pi and helping support other Open Source projects to do the same. I maintain OpenFaaS, and it's always had support to run on ARM devices as well as larger on-premises or cloud servers. The team at Equinix Metal gave me complimentary access to large enterprise-grade ARM servers, Ampere even shipped me a server once, which was unfortunately far too loud to leave on in the house.

Then came the M1 chip from Apple, this is probably where interest in ARM turned mainstream, and binaries for common software started appearing on GitHub releases pages, Docker announced a partnership with ARM and AWS released two generations of server CPUs with an ARM architecture, including support for Lambda.

See also: raspberrypi.org: Five years of Raspberry Pi clusters

The Dev Kit offer

The Dev Kit comes in a small black plastic case with a matte finish, it's got 3x USB-A ports on the back, 2x USB-C on the side and a mini Display Port connector. Inside there's a 8-core Snapdragon 8c 64-bit ARM processor, 32GB of RAM and a 512GB NVMe.


Linux on Microsoft Dev Kit 2023

> "OpenBSD Dev Kit 2023" by Patrick Wildt

Why does that look like such good value? Well there really aren't any other computers in this price range that have a 64-bit ARM processor, and the Raspberry Pi 4 is miles slower in comparison.

> Some comparisons on options for ARM64 compute. https://t.co/9mmL84m4tr pic.twitter.com/T4LIioxEo0
>
> — Alex Ellis (@alexellisuk) October 25, 2022

Comparing the Raspberry Pi 4 8GB against some more expensive machines.

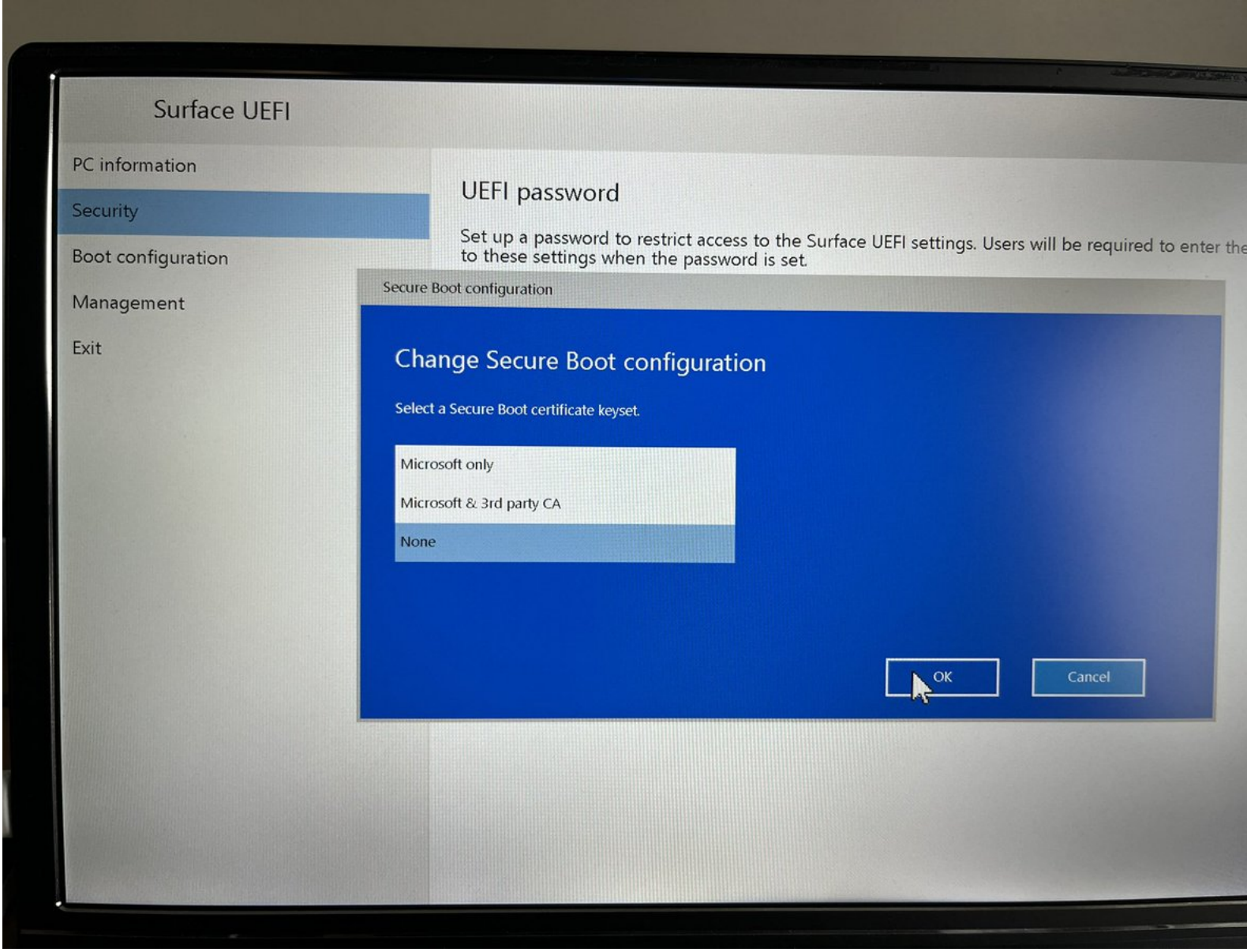
What about the Mac Mini M1 2020? Despite being over 2 years old, it still costs 899 GBP and comes with half the RAM and half the storage. That makes the Dev Kit great value at 570 GBP.

One thing we do know about the Mac Mini is that it's worse value, but a good performer, and Asahi Linux seems relatively stable. It even runs Firecracker.

> The Mac Mini M1 with Ashai Linux installed really is quite fast for the money
>
> Here it is compared to a Raspberry Pi 4 building a custom Linux Kernel.
>
> The Ampere Altra is much faster, even with only 30/80 of the cores allocated to the build. pic.twitter.com/fKqDhlfZcX
>
> — Alex Ellis (@alexellisuk) October 26, 2022

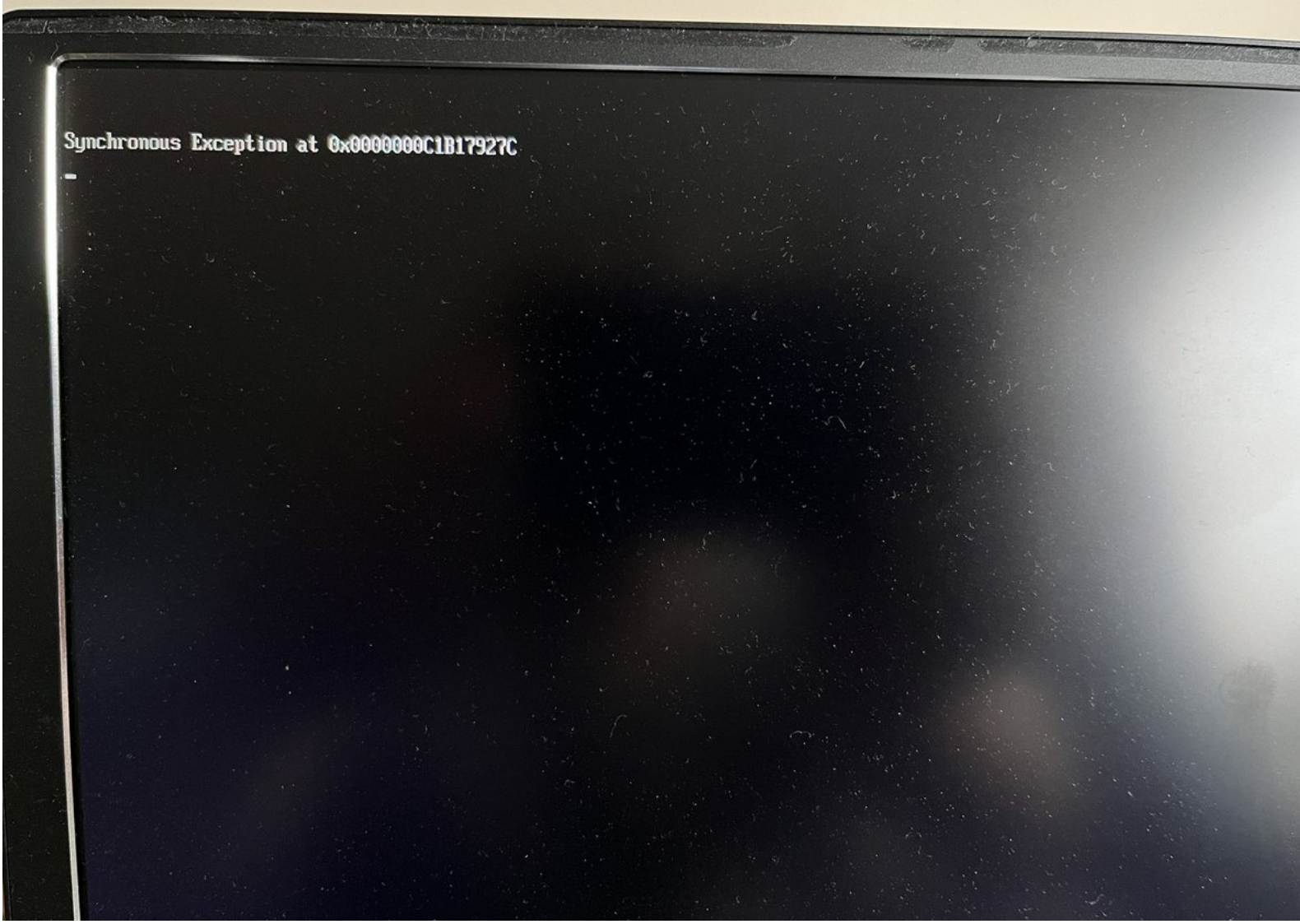
What I've tried

The first thing I figured out was how to disable secure boot.



Disable secure boot by holding the power button and small circular button down and powering up.

That was painless, then I flashed my trusty USB pen drive with Ubuntu 22.04 and held the two larger buttons to "Boot from USB"

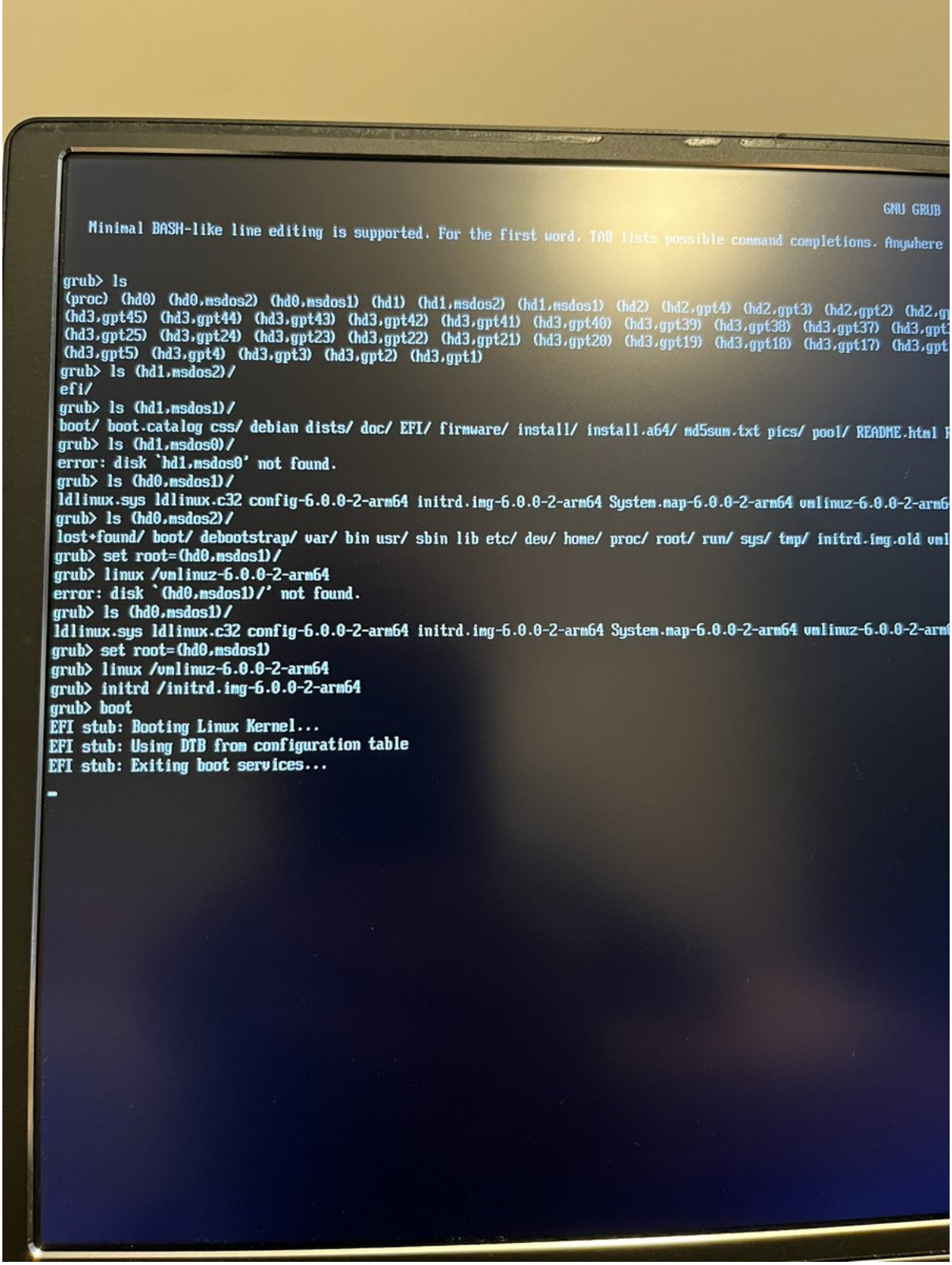


No Linux for you

Grub showed its boot menu, and I thought I'd got it sorted. I couldn't have been more wrong.

So I did what any sane person would do next, try a newer Ubuntu version - 22.10. It broke in exactly the same way.

Now support for the Snapdragon 8c was merged into Linux v6.0, so I thought perhaps I needed to build a new Kernel?



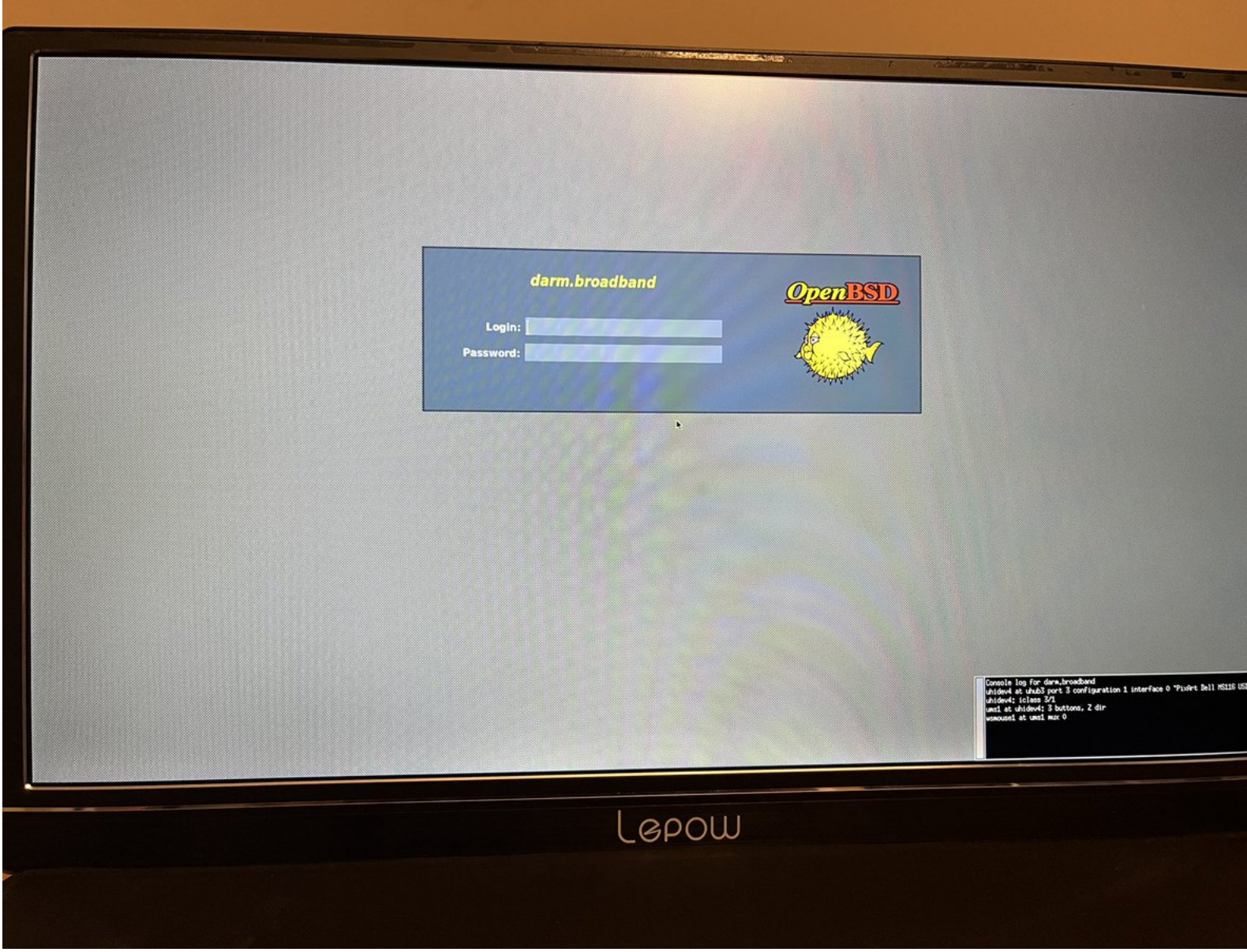
Not booting a v6.0 Kernel

Well that didn't work either.

So I thought maybe I had built the Kernel wrong, and would use a nightly snapshot of Debian sid, which had a v6.0 Kernel built in. That also didn't work.

I spent a day building and rebuilding USB drives and running debootstrap, hoping that one variation or changed Kernel configuration setting would get me booted at least into the Kernel's start-up screen.

Patrick Wildt, an OpenBSD maintainer replied to one of my tweets and told me he had OpenBSD 7.2 up and running on his, so I thought I would at least try that out.



OpenBSD running!

I was able to install Golang (Go) and port some of my own software over ([inlets/mixctl](https://github.com/inlets/mixctl/ - a TCP load-balancer written in Go):

```
darm# pwd
/root/mixctl
darm# ls
.DEREK.yml         .gitignore         Makefile           go.mod             main.go
.git               EULA.md            README.md          go.sum             mixctl
.github            LICENSE            arkade             hack               rules.example.yaml
darm# git remote -v
origin  https://github.com/inlets/mixctl (fetch)
origin  https://github.com/inlets/mixctl (push)
darm# go build
darm# cat rules.example.yaml
version: 0.1

rules:
- name: rpi-k3s
  from: 127.0.0.1:6443
  to:
    - 192.168.1.19:6443
    - 192.168.1.21:6443
    - 192.168.1.20:6443

- name: rpi-ssh
  from: 127.0.0.1:22222
  to:
    - 192.168.1.19:22
    - 192.168.1.21:22
    - 192.168.1.20:22

- name: remap-local-ssh-port
  from: 127.0.0.1:2222
  to:
    - 127.0.0.1:22
darm# ./mixctl -f rules.example.yaml
Starting mixctl by https://inlets.dev/

Forward (rpi-k3s) from: 127.0.0.1:6443 to: [192.168.1.19:6443 192.168.1.21:6443 192.168.1.20:6443]
Forward (rpi-ssh) from: 127.0.0.1:22222 to: [192.168.1.19:22 192.168.1.21:22 192.168.1.20:22]
Forward (remap-local-ssh-port) from: 127.0.0.1:2222 to: [127.0.0.1:22]

Listening on: 127.0.0.1:2222
Listening on: 127.0.0.1:6443
Listening on: 127.0.0.1:22222
^Z[1] + Suspended            ./mixctl -f rules.example.yaml
darm# bg
[1] ./mixctl -f rules.example.yaml
darm# curl -k https://127.0.0.1:6443
2022/10/27 22:00:18 Connected 127.0.0.1:6443 => 192.168.1.21:6443 (127.0.0.1:5207)
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {},
  "status": "Failure",
  "message": "Unauthorized",
  "reason": "Unauthorized",
  "code": 401
}2022/10/27 22:00:18 Closed 127.0.0.1:6443 => 192.168.1.21:6443 (127.0.0.1:5207)
darm# curl -k https://127.0.0.1:6443
2022/10/27 22:00:21 Connected 127.0.0.1:6443 => 192.168.1.20:6443 (127.0.0.1:48316)
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {},
  "status": "Failure",
  "message": "Unauthorized",
  "reason": "Unauthorized",
  "code": 401
}2022/10/27 22:00:21 Closed 127.0.0.1:6443 => 192.168.1.20:6443 (127.0.0.1:48316)
darm#
```

But I need to be able to run Linux, with KVM for this to be useful for my work on actuated - managed and isolated self-hosted CI runners using Firecracker.

I even tried a USB flash drive known to boot with the Lenovo x13s, but it didn't get past Grub.

Not quite giving up

I've used Linux on the desktop since 2018 and it suits my current work very well to be able to run containers directly on the host I'm using, to have a lightweight and responsive UI and the ease of configuration.

Whilst all the various OSS projects I release all have Windows binaries, and it's a decent Operating System, I don't want to have to contend with it on a daily basis.

So I thought, if I can't run Linux on the machine directly, what if I could use WSL2?

> Windows Dev Kit 2023 - Linux (WSL) vs Mac Mini with Asahi Linux
>
> The difference isn't as marked as you would have thought.
>
> I tested with Geekbench 5, hdparm and dd.
>
> But.. pic.twitter.com/3E6jnFbNwk
>
> — Alex Ellis (@alexellisuk) October 31, 2022

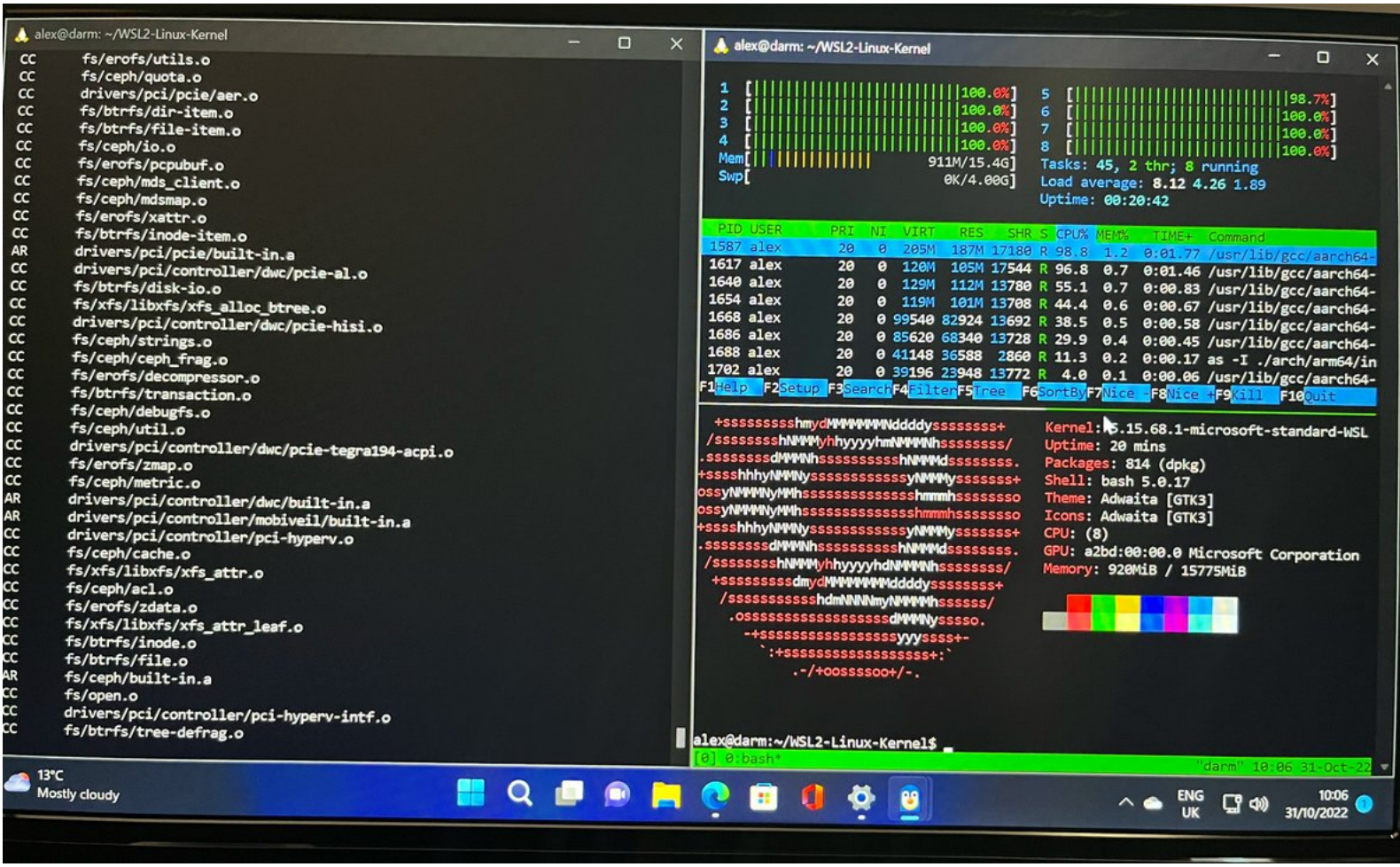
WSL started really quite quickly and so I ran Geekbench 5, hdparm and dd to test the CPU/memory, along with the read and write speed of the disk.

The single-core speed was better, whilst the multi-core speed had dropped off a little compared to running Geekbench directly in Windows 11.

Now because I needed KVM to run Firecracker, I typed in "cpu-checker" and to my dismay, I saw the module hadn't been made available in this Kernel.

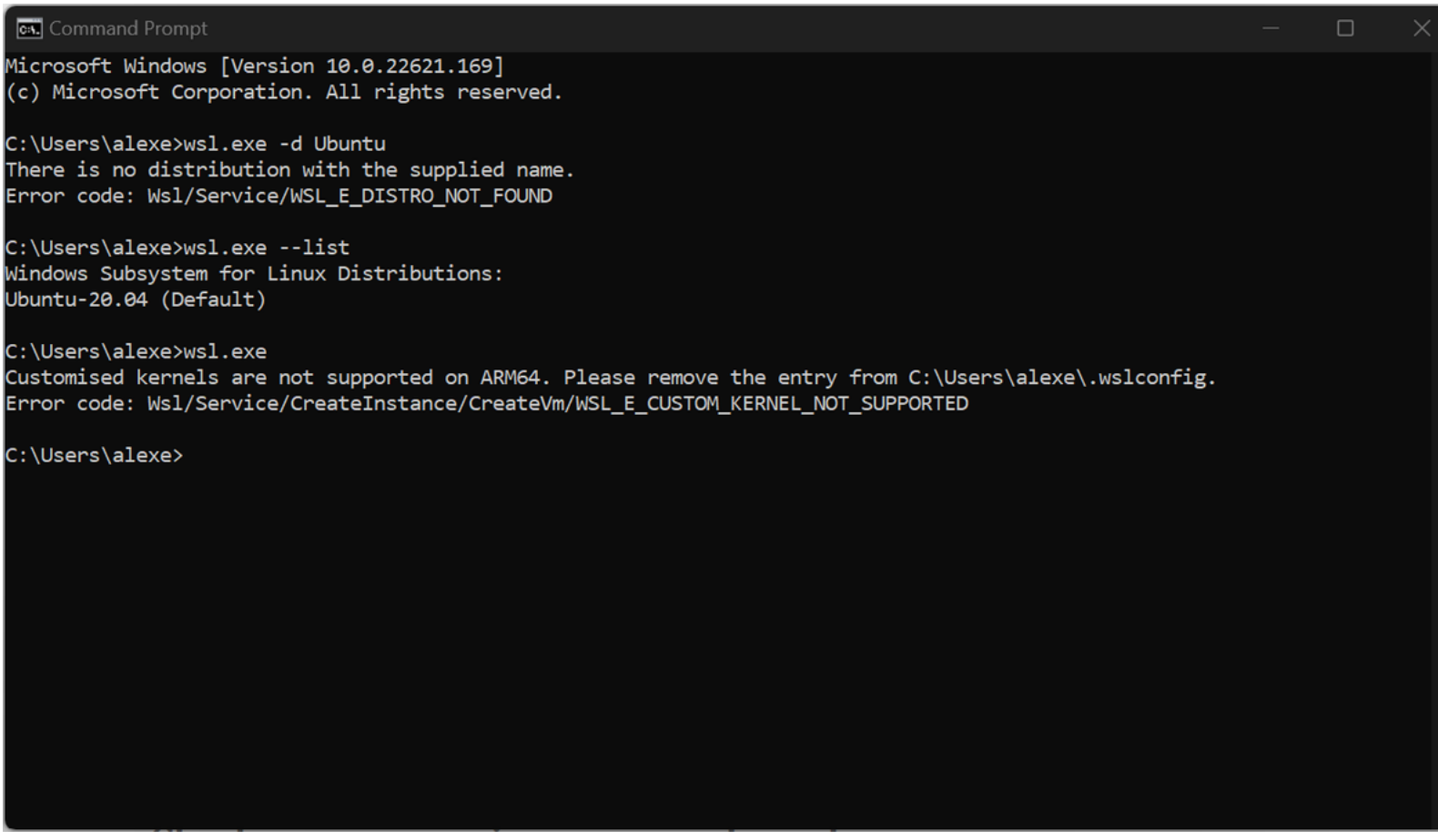
At OpenFaaS Ltd, we've been building Kernels to run in guest VMs using Firecracker for actuated CI runners, for ARM and Intel processors, so it's something I'm very familiar with.

Hayden Barnes has written about how to build and install a custom Kernel for WSL, so I took his instructions and updated them for ARM.



Building my Kernel with KVM enabled

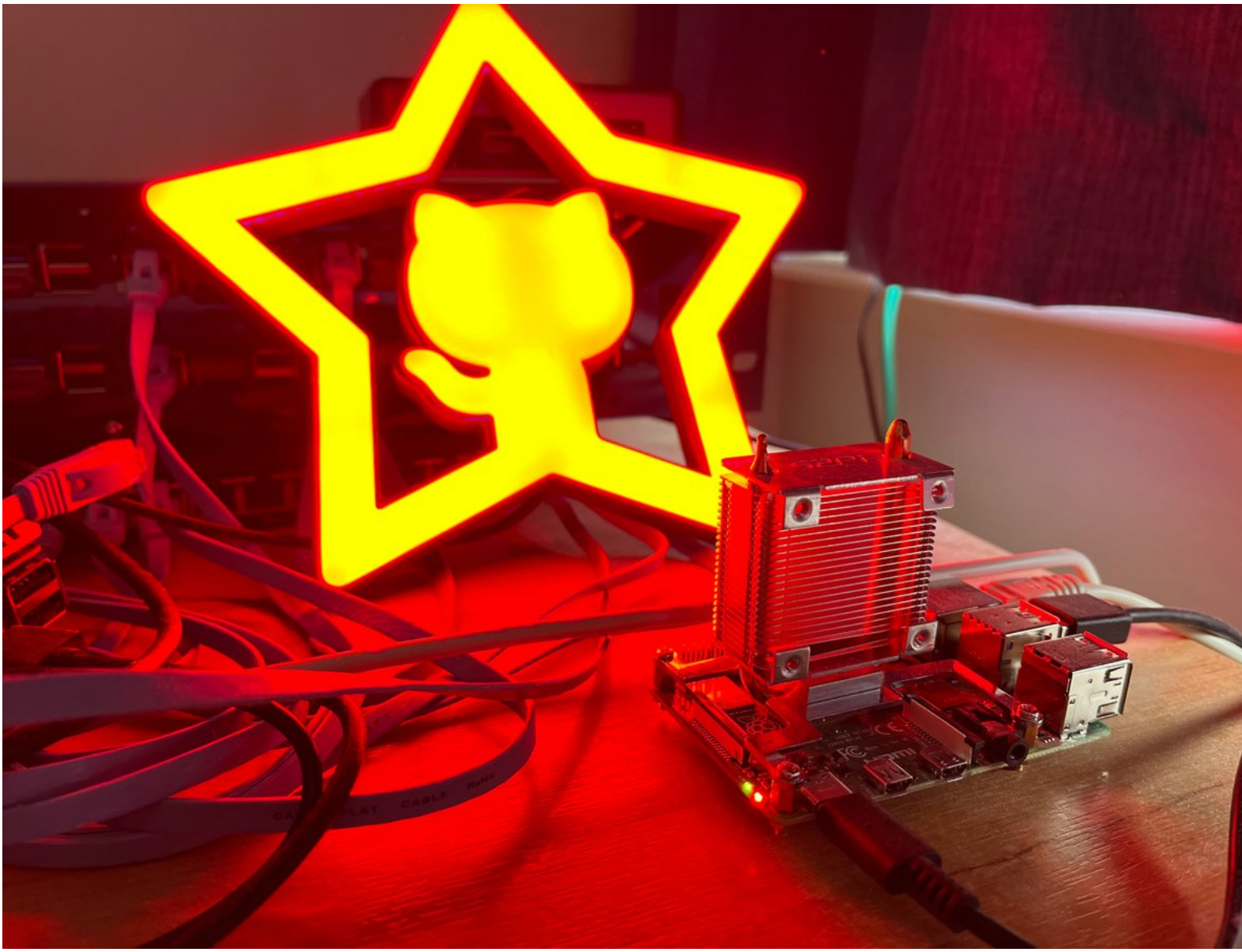
Then just like all my other experiments with trying to get Linux to work how I needed, it felt flat on its face:



For whatever reason, KVM isn't built into the Kernel either as a built-in module or as a loadable module, and a custom Kernel isn't yet supported for ARM64.

Why we need ARM for CI and how my Raspberry Pi beat GitHub's hosted runner

Let's take a quick look at why access to real ARM hardware is important vs. emulation with QEMU.



A Raspberry Pi 8GB kitted out with an NVMe, bought before the global shortage

A start-up founder in Berlin tweeted complaining that his tests were taking 33m30s to execute against an ARM64 CPU profile.

> Does anyone have a @github actions self-hosted runner manifest for me to throw at a @kubernetesio cluster? I'm tired of waiting for emulated arm64 CI runs taking ages.
>
> — Frederic Branczyk (@fredbrancz) October 19, 2022

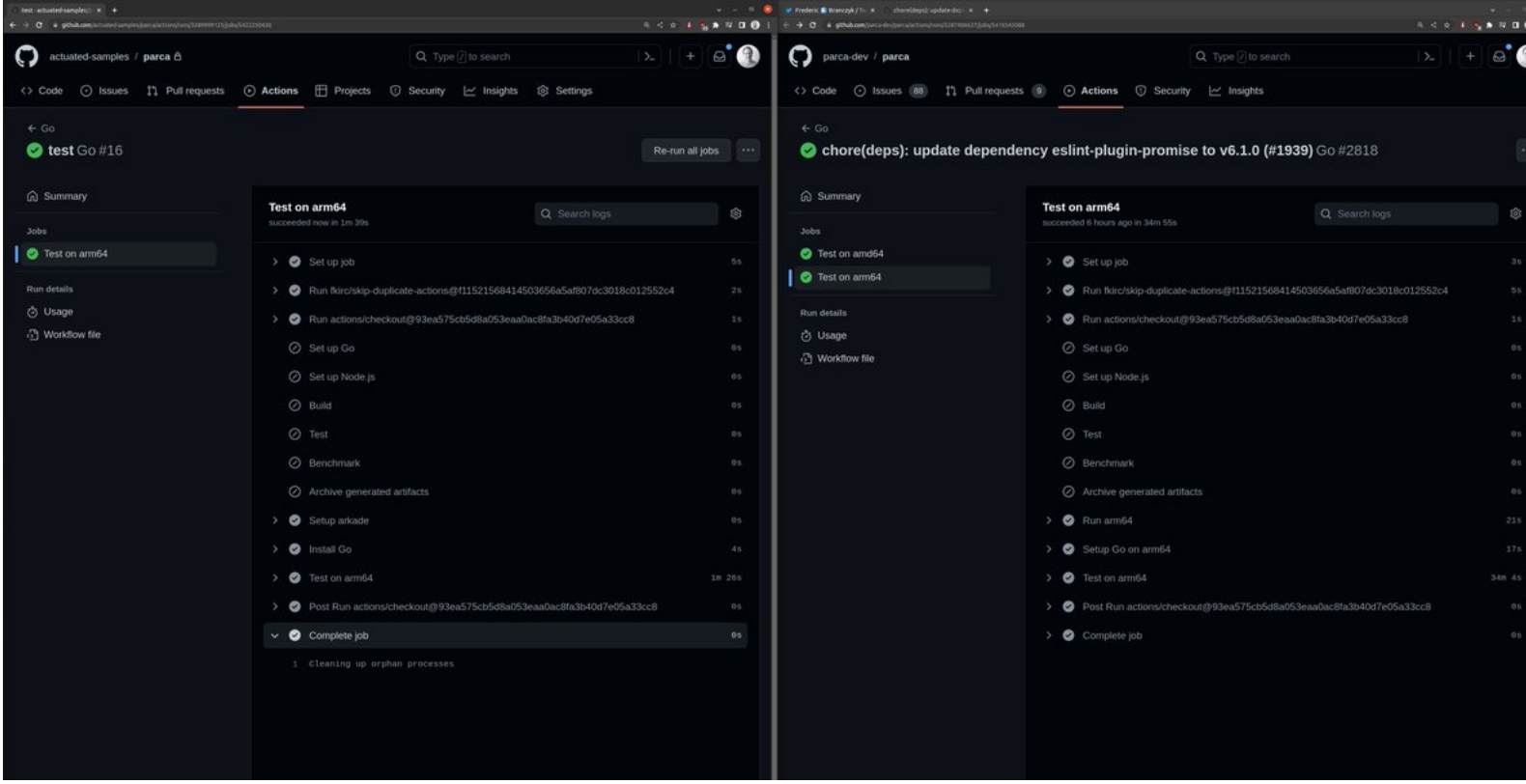
I took a look at his builds and quickly found out that the reason for the slowness was user-space emulation that his team had set up with the "qemu" tool.

qemu is what allows us to create multi-arch builds from a PC that work on ARM computers. But it can also run programs and operating systems.

I cloned his Parca project, then moved it to an organisation where I'd set up actuated, and changed `ubuntu-latest` to `actuated-aarch64`

The first build went onto an ARM64 host at Equinix Metal costing 2.5 USD / hour. The build usually took over 33 minutes on a Hosted Runner without emulation, it took just 1min26s on the Equinix Metal host.

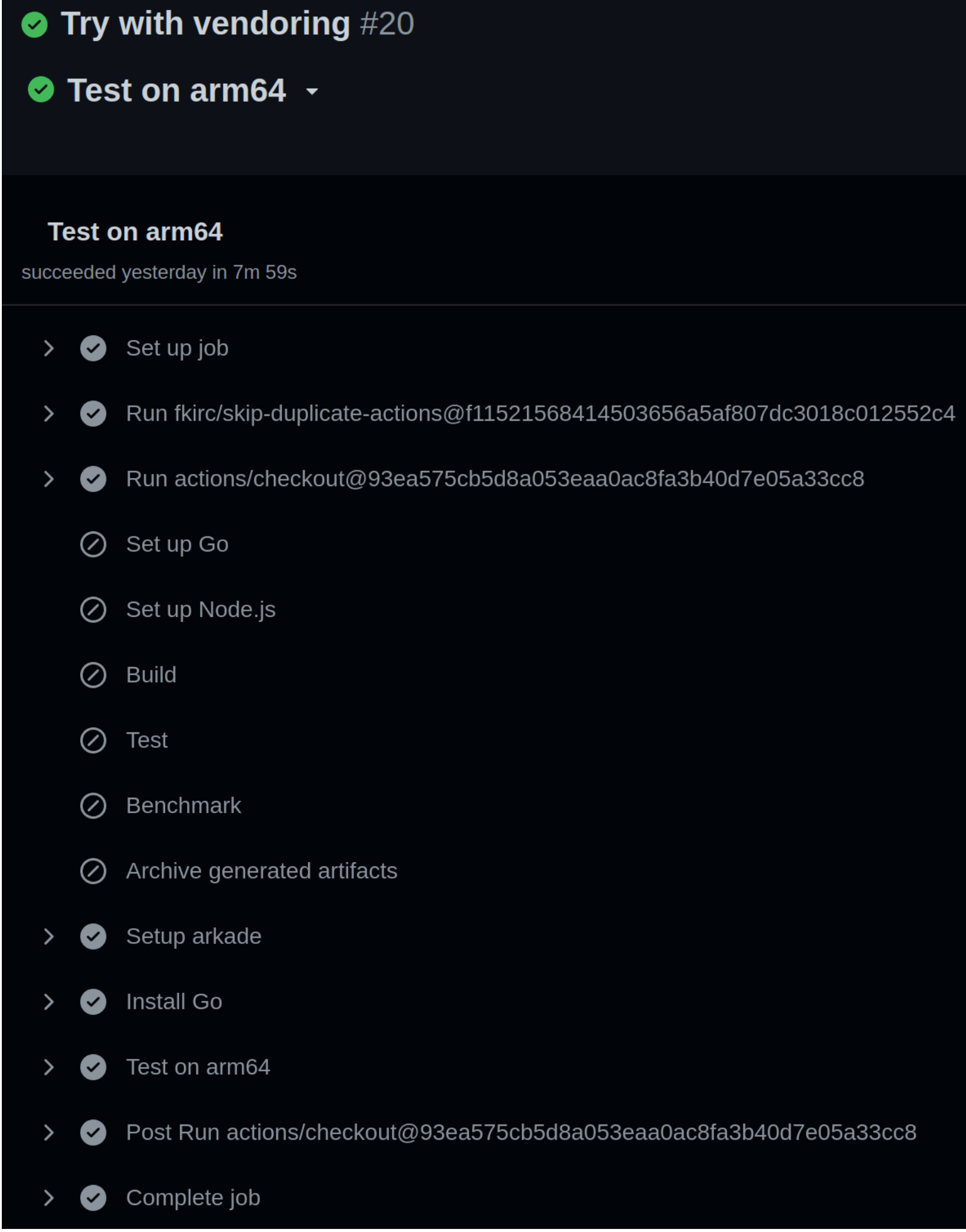
And that was with only 4/80 cores allocated, if it had say 32 allocated, it would have probably completed even quicker.



QEMU vs bare-metal ARM64

Next, I set up the actuated agent on a Raspberry Pi 4. The initial run was 9m30s, 3x faster than emulation, on a device that has a one-time cost of 30-80 GBP total. I then noticed that his build was spending a long time resolving and downloading Go modules. I ran `go mod vendor` and started another build.

That took 7min20s. Saving 2 minutes.



Vendoring to the rescue

So then I looked into what the cheapest ARM64 host would be on the cloud, and it turns out AWS has an a1.metal with 16 cores and 32GB of RAM for 0.48 USD / hour, or 350 USD / mo. So for 350 USD / mo, you can have an ARM64 build-queue 3-4 deep and builds that complete in 1 minute instead of 34.

If some day a Dev Kit 2023 actually works with Linux and KVM, then you could pay for vs an AWS a1.metal instance in just two months.

If you want ARM64 builds or end to end tests for your project, feel free to reach out to me.

Conclusion

The Dev Kit 2023 from Microsoft is a snappy Windows 11 machine with excellent support for Microsoft's WSL2 "Linux experience". WSL2 in this configuration does not support virtualisation or custom Kernel builds. Systemd is turned off by default, which means common software may not work out the box. I didn't test the support, but I was told that you can enable it using these instructions.

When is Linux coming then?

Booting Linux isn't an option at the moment, and may require Microsoft to release a Device Tree Blob (DTB), or a third-party to reverse engineer this. My understanding is that this wasn't required for OpenBSD because it can boot in APCI mode, and the Thinkpad X13s boots because its vendor provided custom DTBs.

Whilst I enjoy this kind of tinkering, it was disappointing that a "Dev Kit", built for developers can neither boot Linux, nor enable a custom Linux Kernel for WSL2.

What's that I hear Hacker News and Reddit cry? "It's a Dev Kit for Windows, you moron!" That may be the case, but Microsoft "Loves Linux", and clearly has worked hard to make WSL2 available out of the box on these devices.

If you're looking for a step above the Raspberry Pi B 8GB for running headless Linux, the 2020 Mac Mini, configured with 16GB of RAM and 256GB of disk space runs to ~ 899 GBP. I wish it were cheaper, but with Ashai Linux it runs Firecracker, KVM, Docker and just about anything else I've thrown at it.

If you get further than I did, please feel free to reach out. For my use-case of building Kubernetes clusters, and supporting Open Source projects and companies to build on ARM64, headless use is absolutely fine.

See also:

- The Past, Present, and Future of Kubernetes on Raspberry Pi - Alex Ellis - KubeCon
- State of netbooting Raspberry Pi in 2021
- Walk-through — install Kubernetes to your Raspberry Pi in 15 minutes

Thanks to:

- Patrick Wildt for encouraging me to try out OpenBSD
- Lucas Lombard for giving me access to his 2020 Mac Mini M1 for testing out actuated

Check out the new thing I'm doing with ARM

With actuated, we're trying to make CI faster, more secure, and isolated whilst taking away a lot of the management and common issues.

Our actuated solution is primarily for Intel/AMD users, but also supports ARM runners.

Feel free to check out the docs, or watch a quick demo of it in action launching Firecracker VMs for each CI job: