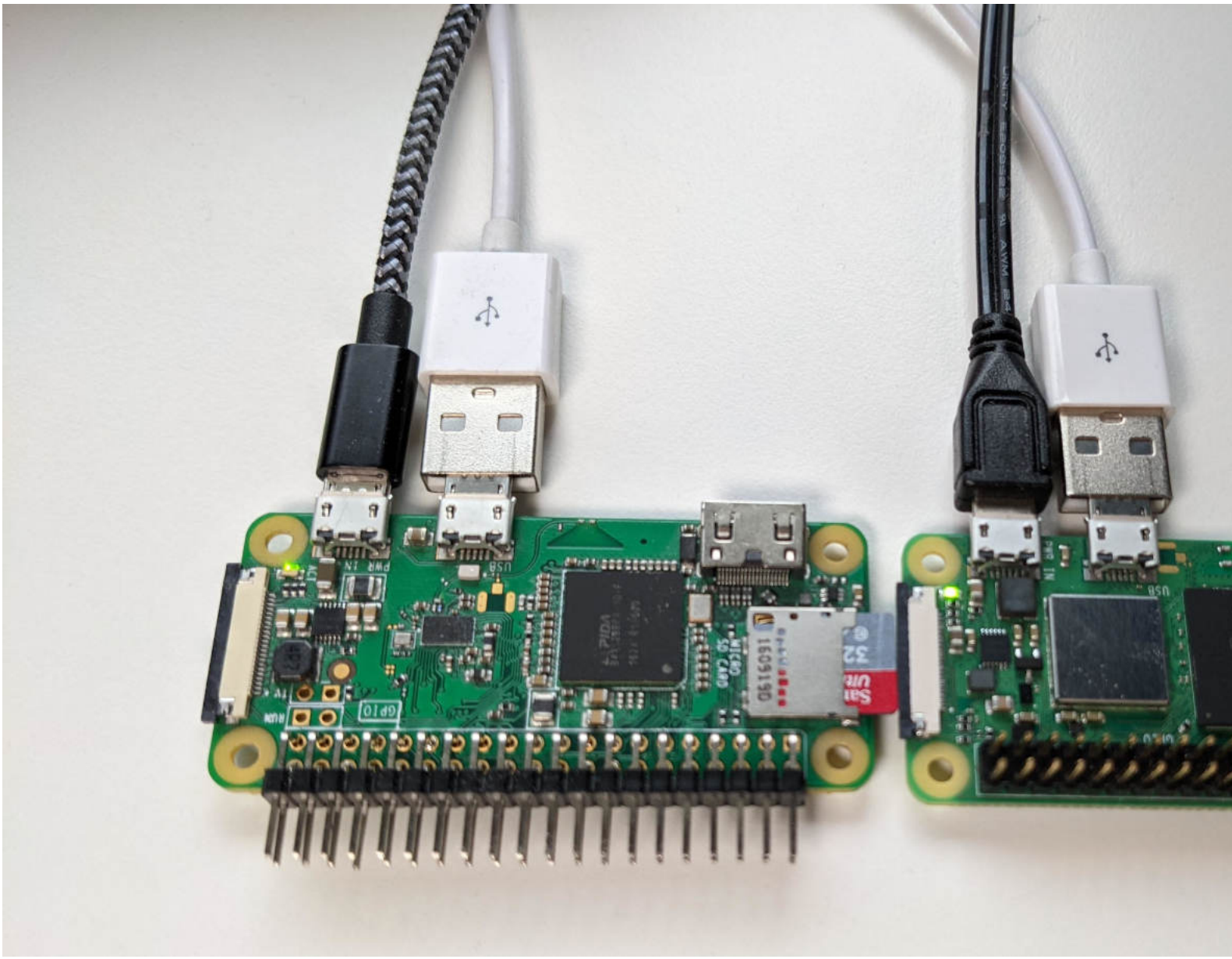
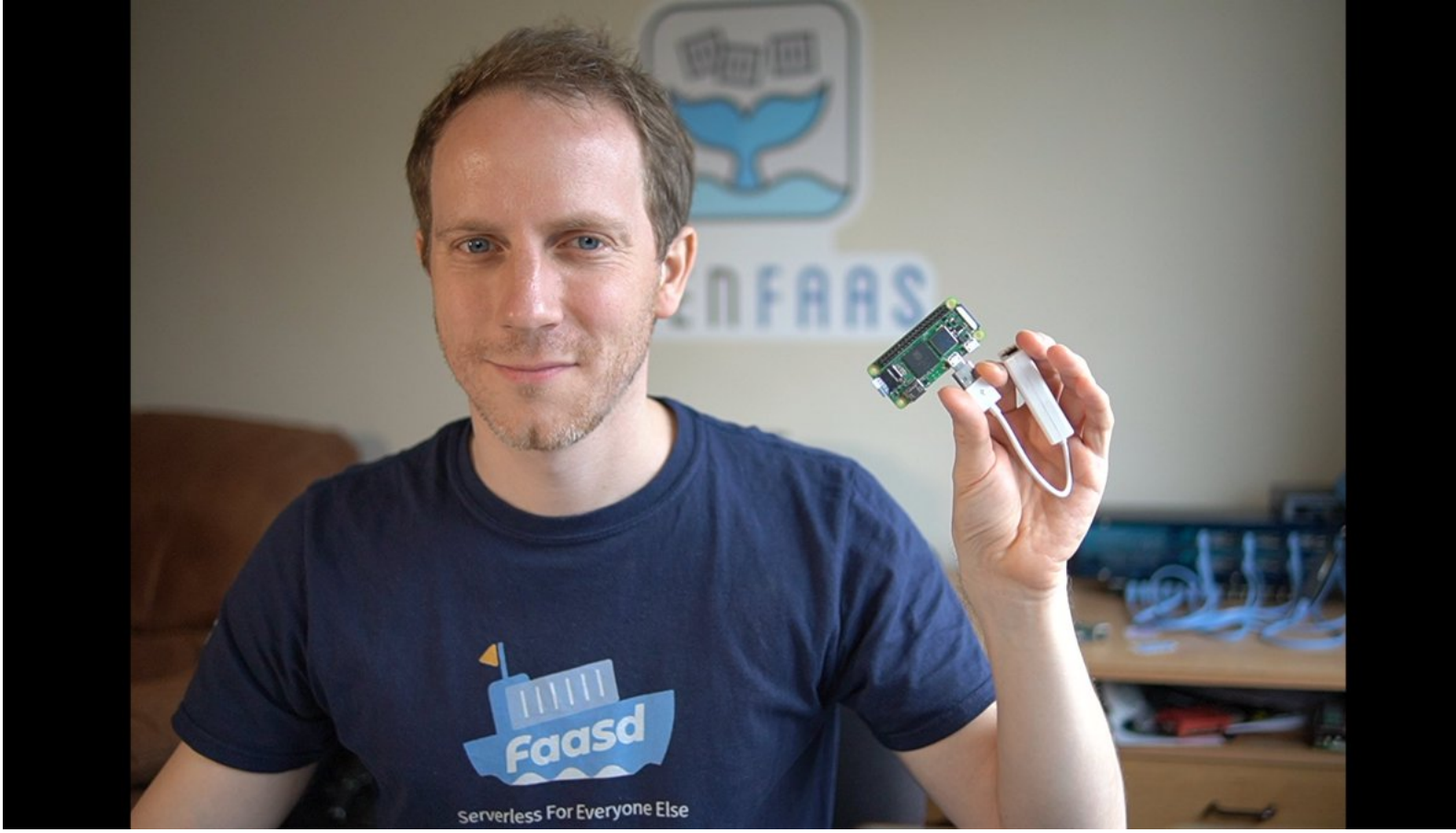
First Impressions with the Raspberry Pi Zero 2 W

This article was fetched from an rss feed(28/10/2021)



Today the Raspberry Pi Foundation released the Raspberry Pi Zero 2 W. I'm going to share my thoughts and experiences with it. Can you run containers? Can you run OpenFaaS? What about K3s? Is it worth buying or will it gather dust?



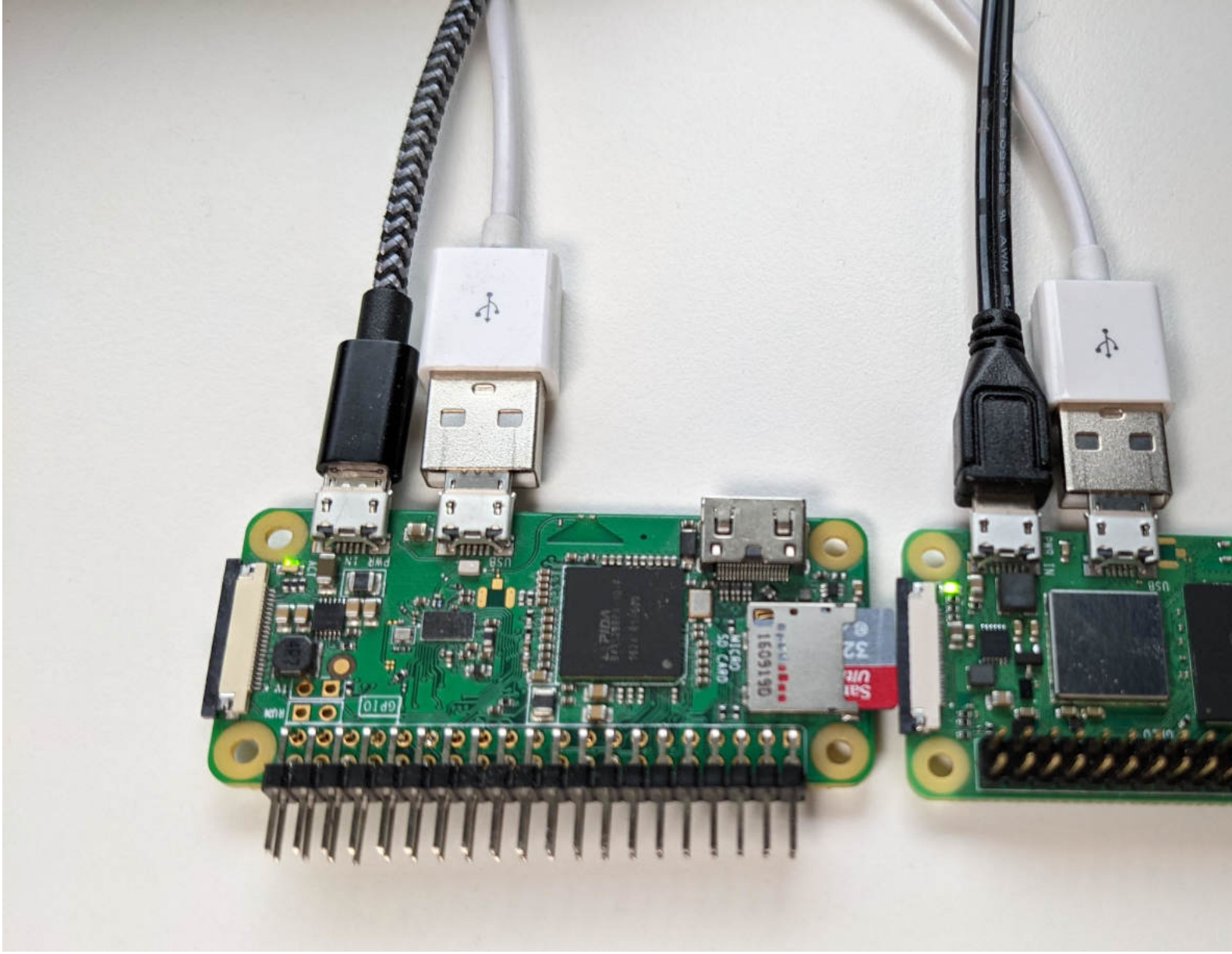
> I've got my faasd t-shirt on and you'll find out why below.

The Raspberry Pi Foundation sent me one of these boards for testing earlier in the year. The original Zero / Zero W had a CPU that was only compatible with armv6 architectures and became quickly underpowered for my Node.js and Go applications.

> Note: this blog post is best viewed on a PC, some images will not show if you're on a mobile device

What's new?

It's been several years since I first reviewed the original Raspberry Pi Zero W, so what's changed and what's it good for?



Twinning - on the left, the original and on the right the new Zero 2.

The new version has the same processor as the original Raspberry Pi 3. From the official press release:

> Priced at $15, Raspberry Pi Zero 2 W uses the same Broadcom BCM2710A1 SoC die as the launch version of Raspberry Pi 3, with Arm cores slightly down-clocked to 1GHz, bundled into a single space-saving package alongside 512MB of LPDDR2 SDRAM. The exact performance uplift over Zero varies across workloads, but for multi-threaded sysbench it is almost exactly five times faster.

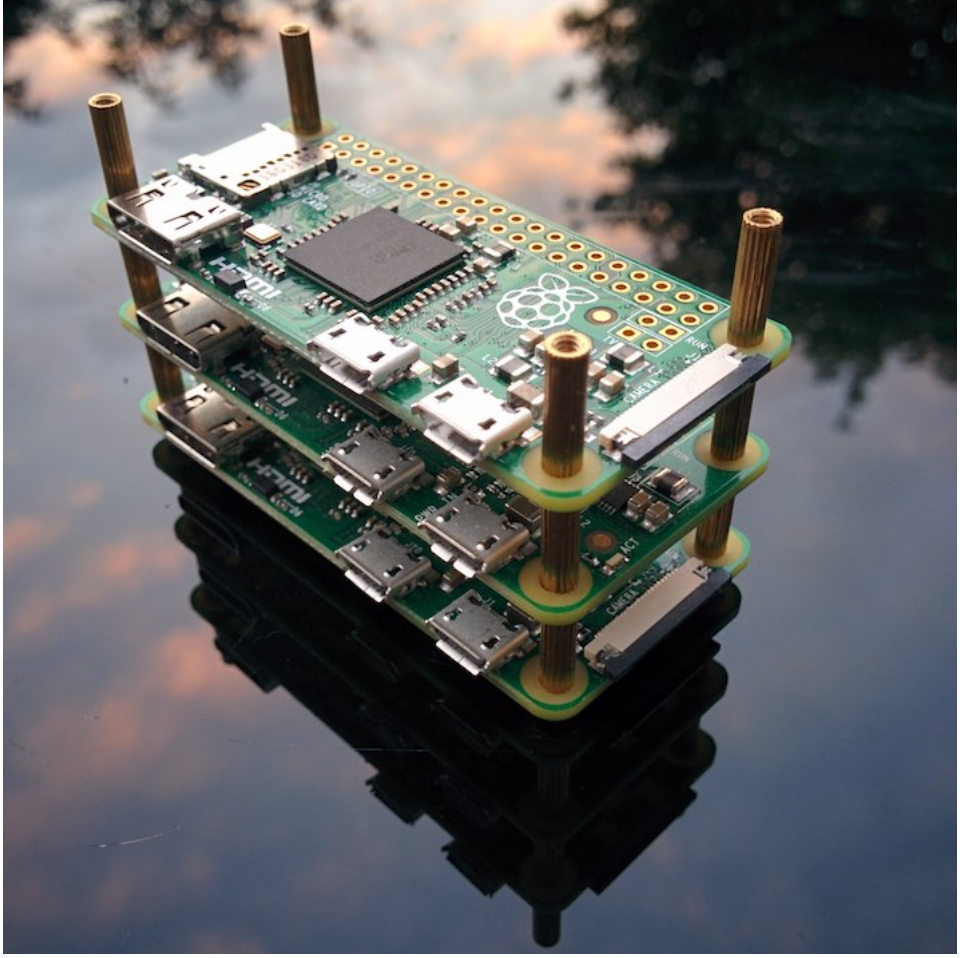
Unfortunately the team were unable to upgrade the RAM to 1GB, which would have made this a smaller Raspberry Pi 3. Yes there are a lot of variations of the Raspberry Pi, so where does this fit?

As I mentioned, running Node.js used to take several seconds to launch, and compiling Go programs could take minutes, where as on a Raspberry Pi 4 these tasks would complete much quicker. This meant that the Raspberry Pi was often last picked for new projects, especially when I wanted to use infrastructure projects.

Over time, many open source projects (including OpenFaaS that I maintain) dropped support for its ARMv6 processor due to its slow speed and issues in the ecosystem.

That lead me to ask the question of what to do with my original Raspberry Pi Zeros. I'm going to give you a quick refresher on what they do well, before getting hands on with the newer version and answering the questions that I know you have.

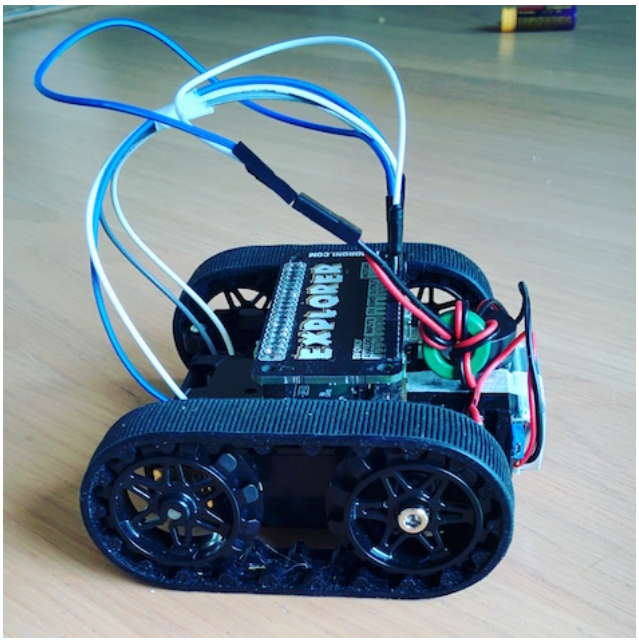
What do we do with all these slow RPi Zeros?



A stack of the original Raspberry Pi Zeros

I remember meeting up with the founders of BitScope and we were on our way to see Eben Upton, wondering what we could do with all the Raspberry Pi Zeros that we had in our collections.

We couldn't use them for clustering because they were too slow, and support was deprecated in many infrastructure projects, sometimes unintentionally.



My ZumoPi robot commissioned by Pimoroni and Linux User and Developer Magazine

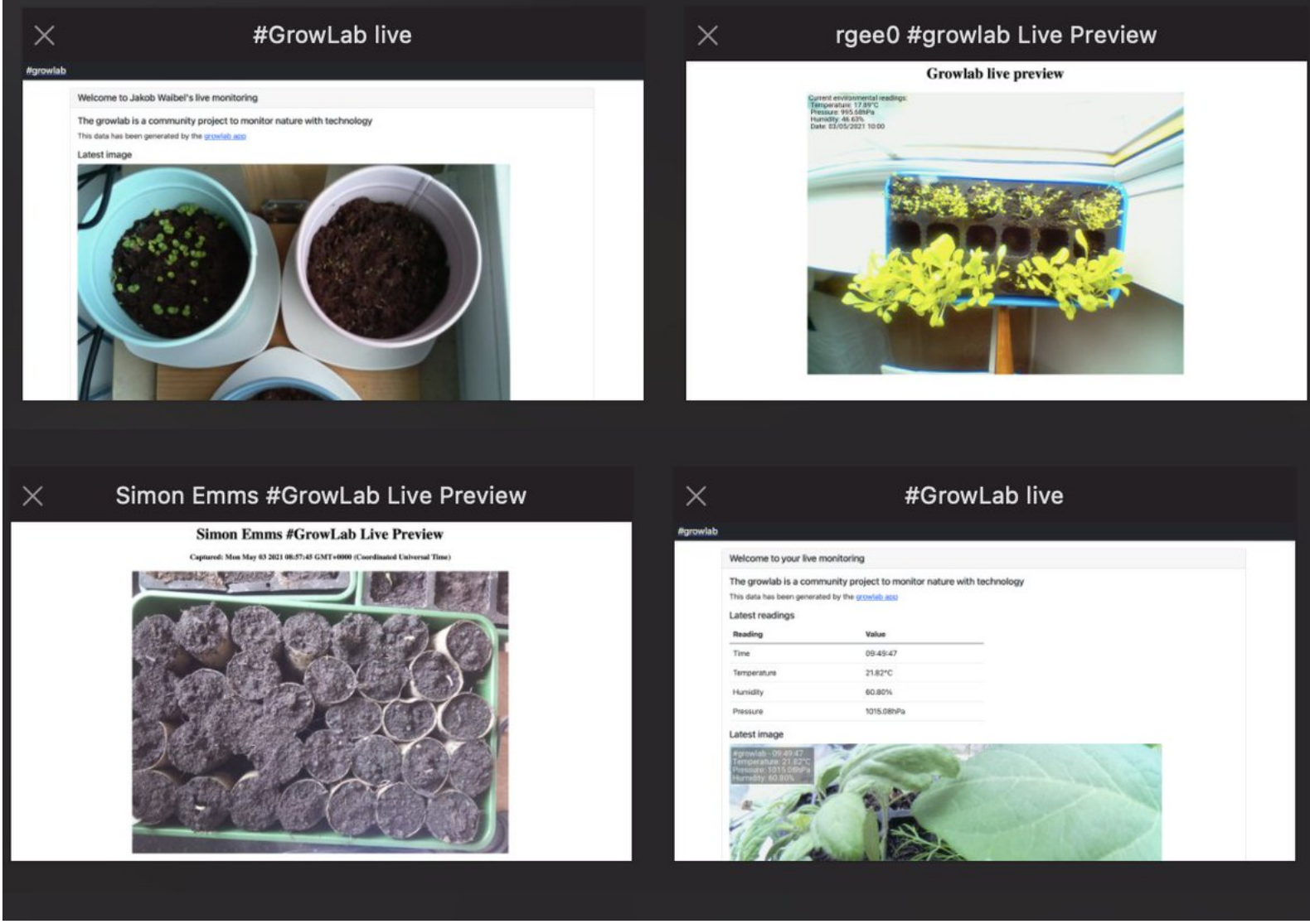
Where they had really shined was in IoT and robotics projects where space and power were on a premium.

My growlab build was the first, but it inspired 20 others to build their own and join me

This year I launched the growlab project to help people all around the world build a lab to grow and monitor seeds and plants. The original Raspberry Pi Zero was a perfect fit here due to its size, cost and plethera of interfacing options: from 40 GPIO pins, to add-ons and a camera port. Its built-in WiFi adapter meant I could place a growlab anywhere with power.

The fruits of my labour, but no Raspberries this year!

The Raspberry Pi Zeros had a temperature sensor connected to them and a wide-angle camera, so you could capture images like this and upload them to a static webpage to share your progress with the community.



Community images from the growlab project

Next, I wanted to aggregate the sensor data in a time-series database. My tool of choice for microservice metrics is Prometheus, but for sensor data InfluxDB is better suited with long term storage and retention policies. It's also fairly light weight. Even so, it could not really fit on my Raspberry Pi along with Grafana, a tool that is often used to visualise data.

So I set up a Raspberry Pi 4 with 2GB of RAM and used the faasd project to host Grafana, InfluxDB and a couple of functions for ingesting data.

The following app runs on the Raspberry Pi Zero: sender/main.py

And then you run this function on the RPi 4 with OpenFaaS: submit-sample/handler.py

The result is that you can then get a beautiful set of graphs showing the environmental data:

I had insulated my shed and installed a Raspberry Pi Zero with a sensor running the sender/main.py code. It was *very very* hot in there.

You can see how the functions and faasd setup works here: Data Logger with faasd

All the code for the growlab projects are available on GitHub

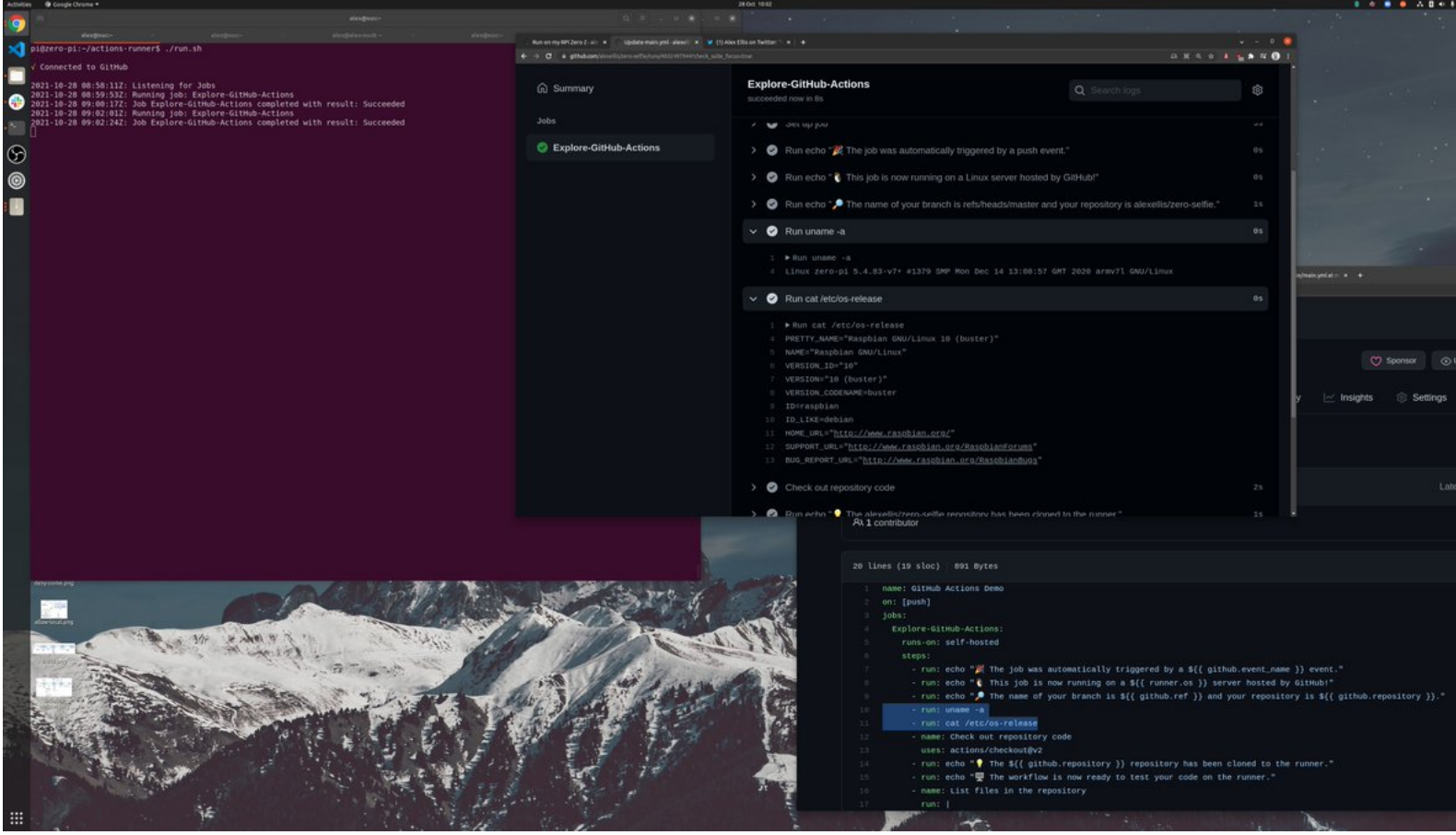
You can find out more about growlab here: https://growlab.dev and in my video presentation from Equinix Metal's GIFEE event

Enter the RPi Zero 2 W

We can actually run containers directly on the Raspberry Pi Zero 2 W, because it supports the ARMv7 architecture, which has much better support across the board.

Self-hosted GitHub Actions runners

Even GitHub provide a self-hosted actions runner for ARMv7, and I tried it out this morning:



See the steps are running on my new RPi Zero 2 in my house

Why would this be useful? Well I guess that the GitHub Actions team saw their product as a CI tool for building binaries, but it has become much more and we don't have time to go into that here. Needless to say, it's become more of a general purpose task runner and a scaleable solution for running batch jobs and extending the GitHub platform.

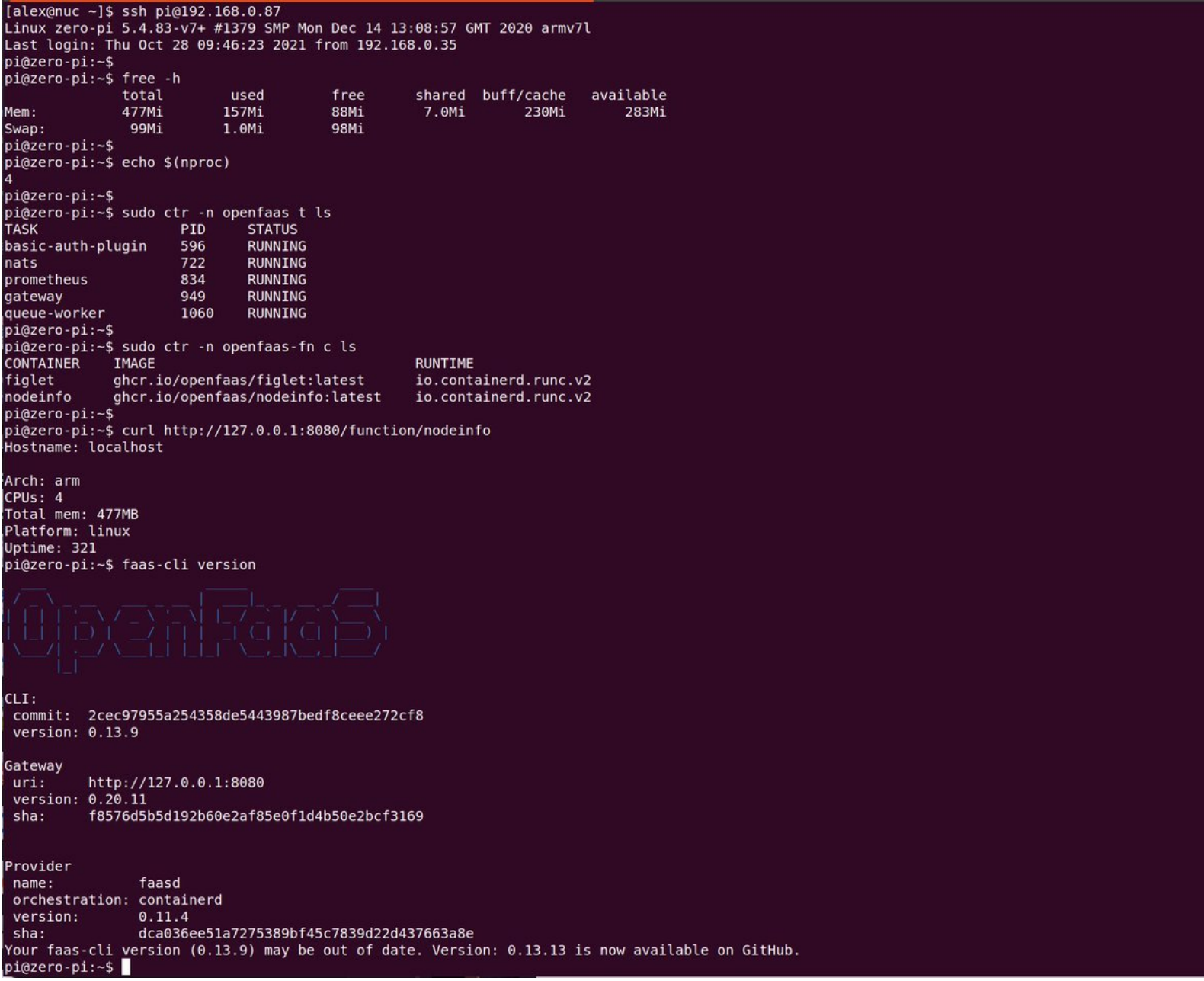
At the beginning of the post I showed you a picture of me wearing my faasd t-shirt and here's why.

Serverless Raspberries

Earlier in the year, I tried really hard to rebuild OpenFaaS, containerd, CNI and all the various components needed as armv6 containers and in the end I had to give up and admit defeat. It was just too slow and too much work, for no real reward.

You can read about how far I got here: [One last trip down memory lane with the Raspberry Pi Zero]

faasd always worked on the Raspberry Pi 3, and now that the two shared the same CPU, it meant I could probably run faasd on this tiny device again.



A screenshot showing containerd and OpenFaaS running

You can also install fully-fledged Docker on these devices using the well-known bash script: `curl -sLS https://get.docker.com | sh` or by using a package manager.

You'll probably want to adjust your `/boot/config.txt` file to squeeze out a few more MB of usable RAM. Just add `gpu_mem=16` to ring out a few megabytes from the GPU, it won't be needing them when running headless.

So whilst the GitHub Actions Runner dials home, and then receives control messages from GitHub's control-plane, what if you wanted to host some functions or webpages and access them from your Raspberry Pi Zero 2?

Fortunately, inlets has you covered there. Unlike tools like Ngrok, it's actually built for use in production and integrates extremely well with containers.

I ran the inlets-pro client on my computer using a tunnel server that I created on DigitalOcean.

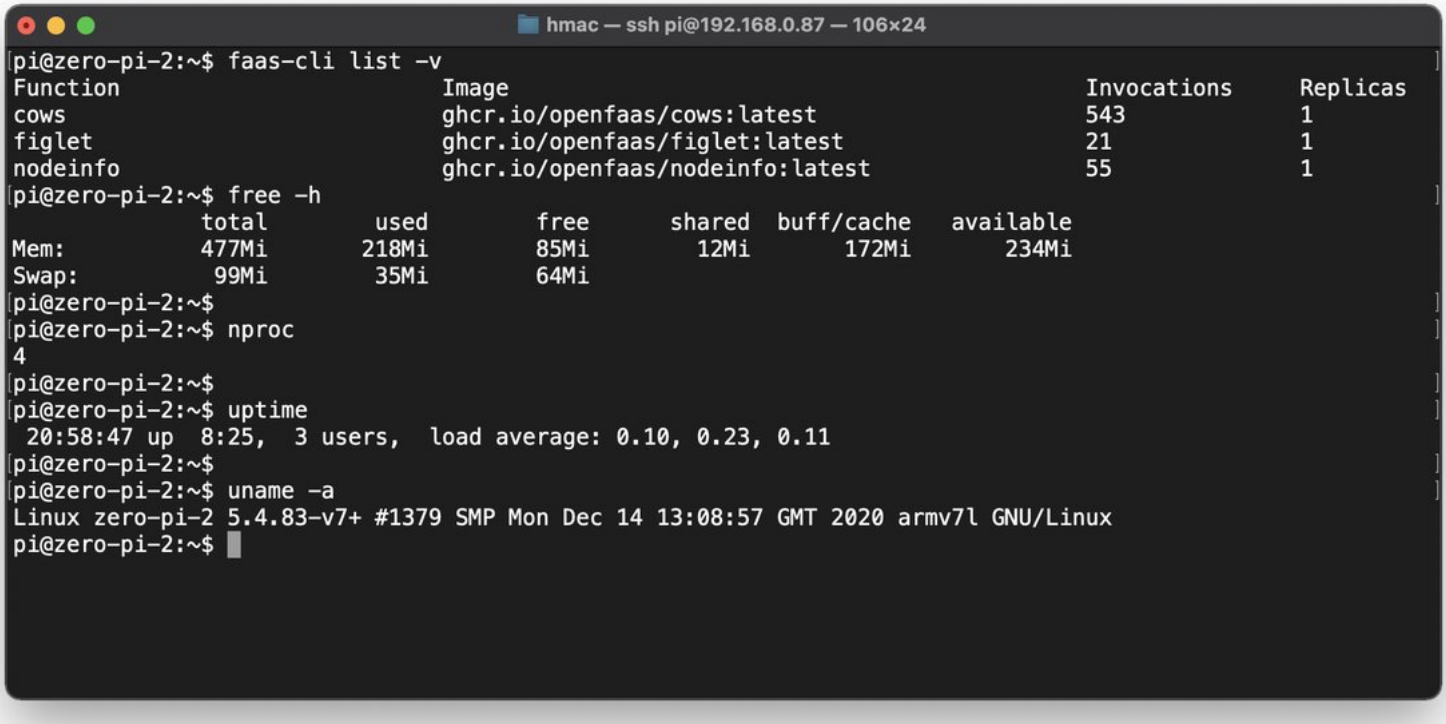
Feel free to try out my sample functions over the tunnel:

```
$ curl -sL https://zero2.exit.o6s.io/function/nodeinfo

Hostname: localhost

Arch: arm
CPUs: 4
Total mem: 477MB
Platform: linux
Uptime: 350

$ curl -sL https://zero2.exit.o6s.io/function/figlet -d `whoami`

  _
 _ __ (_)
| '_ \| |
| |_) | |
| .__/|_|
|_|

$ curl -sL https://zero2.exit.o6s.io/function/cows

        (__)       (----------)
        (--) . . . ( *>YAWN<* )
   /------\/       (----------)
  /|     ||
 * ||----||
Cow after pulling an all-nighter
```

Why not try out the cows function in your browser?

There's templates for building your own functions in Node.js, Go, JavaScript, Bash, Python and other languages too. See an example of each on the homepage.

We've also published a set of multi-arch functions to the store which you can see via `faas-cli store list`. Here's the store's GitHub repo to see how they work with common CLIs like `curl`, `ffmpeg`, `nslookup`, `hey`, `nmap` and so forth containerised and ready to run on your RPis: openfaas/store-functions



`faas-cli list` shows the amount of invocations so far on the various functions.

Of course OpenFaaS isn't just about having fun. It's got plenty of real-world use-cases, that you can learn about: Exploring Serverless Use-cases from Companies and the Community

I wrote a blog post about setting up faasd on a Raspberry Pi, and toured some of the features like asynchronous invocations and multi-arch container images. It's now much much easier to install, but you may like seeing what else it can do: faasd - lightweight Serverless for your Raspberry Pi

You can set up your own tunnel with custom domains using the guides below:

- Serve traffic through a private self-hosted tunnel

Inlets can also turn your RPi Zero into a handy file sharing server:

- The simple way to share files directly from your computer

To learn how to setup faasd on your Raspberry Pi and build functions with practical examples, try my eBook: Serverless For Everyone Else

What about K3s?

I would not recommend running K3s on a Raspberry Pi 3 due to its well-known limits on I/O. These cause timeouts and break K3s when running with etcd, the tool used to maintain state across the cluster.

If you're only wanting to run a single node Kubernetes cluster, then you may have a better experience with faasd which is free and open source and foregoes any bogging down with clustering.

That said, I thought you'd like to see if it would install.

Running a Kubernetes cluster over WiFi is a terrible idea. I added a USB network adapter for this experiment.

You need to append `cgroup_memory=1 cgroup_enable=memory` to the `/boot/cmdline.txt` file and reboot before attempting this.

k3sup can be used to install K3s via SSH and merge a new context into your KUBECONFIG for use with kubectl. All without logging into the RPi itself.

```
k3sup install \
  --ip 192.168.0.87 \
  --user pi \
  --k3s-channel v1.22 \
  --merge \
  --context rpizero2 \
  --local-path $HOME/.kube/config


Merging with existing kubeconfig at /home/alex/.kube/config
Saving file to: /home/alex/.kube/config

# Test your cluster with:
export KUBECONFIG=/home/alex/.kube/config
```

To my surprised, it actually worked, but seemed to get really bogged down starting up, taking multiple seconds to return `kubectl get nodes`

Shortly after it choked.

In the logs for the k3s service, I saw very high latency from etcd:

```
Oct 28 11:17:56 zero-pi k3s[484]: I1028 11:17:56.769385     484 trace.go:205] Trace[1920479996]: "GuaranteedUpdate etcd3" type:*v1.Endpoints (28-Oct-202
Oct 28 11:17:56 zero-pi k3s[484]: Trace[1920479996]: ---"initial value restored" 196ms (11:17:54.657)
```

```
Oct 28 11:17:56 zero-pi k3s[484]: Trace[1920479996]: ---"Transaction prepared" 1606ms (11:17:56.263)
Oct 28 11:17:56 zero-pi k3s[484]: Trace[1920479996]: ---"Transaction committed" 504ms (11:17:56.768)
```

Followed by `kubectl get nodes` eventually timing out.

```
$ kubectl get node
Unable to connect to the server: net/http: TLS handshake timeout
```

Then I saw hundreds of lines of errors.

It's probably safe to say that whilst this device had no issues running faasd, it's just not suitable for K3s, even with a single node. The limited RAM and I/O is just not a good combination for a container orchestor built to run on much more powerful hardware. Use RPi4s instead.

See also: Self-hosting Kubernetes on your Raspberry Pi

Building a little Go program

This uses Go 1.17 to build arkade which has very few dependencies, and can be used to download CLIs in a similar way to homebrew, but for binaries. It also has app installers for around 40 Kubernetes projects.

```
sudo rm -rf /usr/local/go

sudo apt install -qy git

curl -SL -o go.tgz https://golang.org/dl/go1.17.2.linux-armv6l.tar.gz

sudo mkdir -p /usr/local/go
sudo tar -xvf ./go.tgz -C /usr/local/go --strip-components=1
export PATH=$PATH:/usr/local/go/bin
```

Clone arkade:

```
git clone https://github.com/alexellis/arkade --depth=1
cd arkade

# Resolve and download dependencies according to go.mod
time go mod download

# Build a binary
time go build
```

Here's the original RPi Zero

```
pi@zero-pi-1:~/arkade $ time go mod download

real    5m54.399s
user    3m51.653s
sys     1m23.257s

pi@zero-pi-1:~/arkade $ time go build

real    5m29.314s
user    4m37.119s
sys     0m40.533s
```

And the new model:

```
pi@zero-pi-2:~/arkade$ time go mod download

real    0m44.392s
user    0m40.020s
sys     0m5.177s

pi@zero-pi-2:~/arkade$ time go build

real    0m50.969s
user    1m48.109s
sys     0m10.250s
pi@zero-pi-2:~/arkade$
```

And an RPi 4:

```
pi@k4s-1:~/arkade $ time go mod download

real    0m33.653s
user    0m41.719s
sys     0m10.816s

pi@k4s-1:~/arkade $ time go build

real    0m20.983s
user    0m45.630s
sys     0m8.521s
```

So we have a difference of 5m54.399s vs 0m50.969s for just downloading and resolving a few dependencies from the go.mod file. That is a marked improvement, but still not ideal. Even the RPi 4 took 33 seconds for this operation.

For the build, things were even worse, with the original RPi Zero taking `real 5m29.314s` which is so long that I almost gave up waiting. The RPi Zero 2 completed in `0m50.969s`

Just for fun, I also built arkade on my Apple MBA M1:

```
bash-3.2$ time go mod download

real    0m6.409s
user    0m2.748s
sys     0m1.223s

bash-3.2$ time go build

real    0m0.174s
user    0m0.260s
sys     0m0.286s
```

Even the RPi 4 is relatively sluggish. Who wants to wait 20s between changing code and running it? For me, cross-compiling Go on a faster computer is a better alternative.

Here's a snippet from arkade's Makefile, which is how we cross-compile inside a GitHub Action:

```
        CGO_ENABLED=0 GOOS=linux GOARCH=arm GOARM=6 go build -ldflags $(LDFLAGS) -a -installsuffix cgo -o bin/arkade-armhf
```

That binary can then be copied to the Raspberry Pi using `scp` or downloaded via HTTP endpoint.

Building hello-world instead

Let's compare these results to a trival app I wrote for my eBook Everyday Go. It simply accepts a couple of version flags during a build, then outputs them back to the user at runtime.

```
package main

import "fmt"

var (
        Version   string
        GitCommit string
)

func main() {
        fmt.Printf("release-it: %s, commit: %s\n", Version, GitCommit)
}


git clone https://github.com/alexellis/release-it --depth=1
cd release-it

time go build -mod=vendor
```

RPi Zero 1: `real 0m5.638s`
RPi Zero 2: `real 0m1.622s`
RPi 4: `real 0m0.766s`

So building Go programs is much much faster on the RPi Zero 2, but it's still not responsive enough for me and I would recommend building your code on a regular Intel computer and copying the binary across for testing.

In fact, that's what we do for arkade in the release process which you can see here: arkade/.github/workflows/publish.yml

Before I leave you, let's see how quickly Node.js will launch on the new processor?

```
curl -sLS -o node.tar.xz https://nodejs.org/dist/v16.13.0/node-v16.13.0-linux-armv7l.tar.xz
sudo tar -xvf node.tar.xz -C /usr/local/ --strip-components=1

cat <<EOF > index.js

console.log("Hi " + process.env.USER)

EOF

time node index.js
```

To further put my point home about armv6, the Node.js project has stopped building Node for the original RPi Zero, so I had to use an unofficial build here: `https://unofficial-builds.nodejs.org/download/release/v14.10.0/node-v14.10.0-linux-armv6l.tar.xz`

RPi Zero 1: `real 0m1.551s`
RPi Zero 2: `real 0m0.557s`
RPi 4: `real 0m0.355s`

1.5s to load Node.js, with a program with zero dependencies is far too long. The RPi Zero 2 brings that down to half a second, which I think most people could proably tolerate. This number will be higher if there is more that needs to be loaded in at start-up.

Why is loading Node.js important? I mentioned the GitHub Actions runner earlier, that uses a bundled copy of Node.js to execute some of your workflows.

Wrapping up

Should you buy a Raspberry Pi Zero 2 W?

Building and running microservices and functions in Go and Node.js will be faster. You can run more tools than before like the GitHub Actions self-hosted runner, and various open source projects which only build for ARMv7 CPUS. In short, it will cope better with pretty much anything you throw at it.

If you'd like to run K3s or experiment with containers and clustering, you need a Raspberry Pi 4, ideally with 2GB of RAM or more. I have write up on clustering here: State of netbooting Raspberry Pi in 2021

What about 64-bit OSes like Ubuntu? According to my sources, a 32-bit OS suits this board better, so who are we to argue? I was told that the 64-bit OS is slower for most things compared to a 32-bit OS. Booting the graphical desktop taking an extra second, building the kernel taking ~ 2s more, and launching a webpage in Chrome taking ~ 5s more. Do you really need a 64-bit ARM OS, **even if it's demonstrably slower**? Well, if you're absolutely sure, then an RPi4 may be a better choice.

For faasd and serverless functions, you can try it out on your RPi Zero 2, but the minimum I would recommend is an RPi 3 or 4. You will thank me later, when you can pack more functions in and get a more responsive experience.

For sensors and low-powered operations that may need to run from a battery, the original Raspberry Pi Zero still may be the best option from a power consumption point of view. According to my Kill-O-Watt, my Zero 2 is drawing 4.8W at idle, and around 8.1W during the build of arkade. The original RPi Zero draws 2.3W at idle, around half of that.

If you're reading this, then the chances are that (just like me) you've already ordered one. There are more things you can run on it than with the older generation, I'm sure you'll have fun playing with it.

> What are you going to do with yours? Follow me on Twitter @alexellisuk if you'd like to join the discussion and find out more.

You may also like:

- A Tour of Inlets - A Tunnel Built for the Cloud
- State of netbooting Raspberry Pi in 2021
- Will it cluster? k3s on your Raspberry Pi