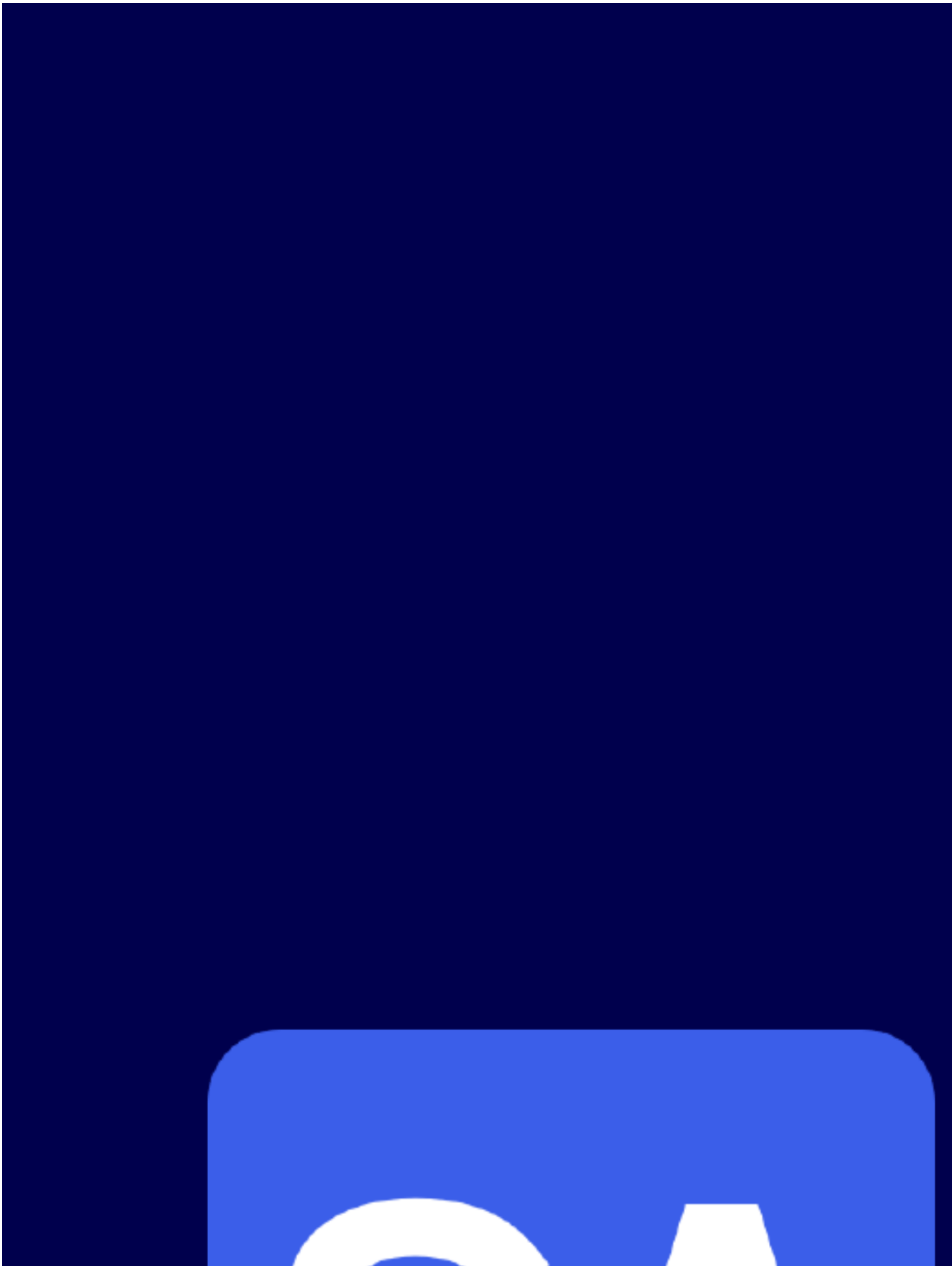
Blazing fast CI with MicroVMs

This article was fetched from an rss feed(10/11/2022)




SA

Around 6-8 months ago I started exploring MicroVMs out of curiosity. Around the same time, I saw an opportunity to **fix** self-hosted runners for GitHub Actions. Actuated is now in pilot and aims to solve most if not all of the friction.

There's three parts to this post:

1. A quick debrief on Firecracker and MicroVMs vs legacy solutions
2. Exploring friction with GitHub Actions from a hosted and self-hosted perspective
3. Blazing fast CI with Actuated, and additional materials for learning more about Firecracker

> We're looking for customers who want to solve the problems explored in this post. Register for the pilot

1. A quick debrief on Firecracker

> Firecracker is an open source virtualization technology that is purpose-built for creating and managing secure, multi-tenant container and function-based services.

I learned about Firecracker mostly by experimentation, building bigger and more useful prototypes. This helped me see what the experience was going to be like for users and the engineers working on a solution. I met others in the community and shared notes with them. Several people asked "Are microVMs the next thing that will replace containers?" I don't think they are, but they are an important tool where hard isolation is necessary.

Over time, one thing became obvious:

> MicroVMs fill a need that legacy VMs and containers can't.

If you'd like to know more about how Firecracker works and how it compares to traditional VMs and Docker, you can replay my deep dive session with Richard Case, Principal Engineer (previously Weaveworks, now at SUSE).

> Join Alex and Richard Case for a cracking time. The pair share what's got them so excited about Firecracker, the kinds of use-cases they see for microVMs, fundamentals of Linux Operating Systems and plenty of demos.

2. So what's wrong with GitHub Actions?

First let me say that I think GitHub Actions is a far better experience than Travis ever was, and we have moved all our CI for OpenFaaS, inlets and actuated to Actions for public and private repos. We've built up a good working knowledge in the community and the company.

I'll split this part into two halves.

What's wrong with hosted runners?

Hosted runners are constrained

Hosted runners are incredibly convenient, and for most of us, that's all we'll ever need, especially for public repositories with fast CI builds.

Friction starts when the 7GB of RAM and 2 cores allocated causes issues for us - like when we're launching a KinD cluster, or trying to run E2E tests and need more power. Running out of disk space is also a common problem when using Docker images.

GitHub recently launched new paid plans to get faster runners, however the costs add up, the more you use them.

What if you could pay a flat fee, or bring your own hardware?

They cannot be used with public repos

From GitHub.com:

> We recommend that you only use self-hosted runners with private repositories. This is because forks of your public repository can potentially run dangerous code on your self-hosted runner machine by creating a pull request that executes the code in a workflow.
>
> This is not an issue with GitHub-hosted runners because each GitHub-hosted runner is always a clean isolated virtual machine, and it is destroyed at the end of the job execution.
>
> Untrusted workflows running on your self-hosted runner pose significant security risks for your machine and network environment, especially if your machine persists its environment between jobs.

Read more about the risks: Self-hosted runner security

Despite a stern warning from GitHub, at least one notable CNCF project runs self-hosted ARM64 runners on public repositories.

On one hand, I don't blame that team, they have no other option if they want to do open source, it means a public repo, which means risking everything knowingly.

Is there another way we can help them?

I spoke to the GitHub Actions engineering team, who told me that using an ephemeral VM and an immutable OS image would solve the concerns.

There's no access to ARM runners

Building with QEMU is incredibly slow as Frederic Branczyk, Co-founder, Polar Signals found out when his Parca project was taking 33m5s to build.

I forked it and changed a line: `runs-on: actuated-aarch64` and reduced the total build time to 1m26s.

> This morning @fredbrancz said that his ARM64 build was taking 33 minutes using QEMU in a GitHub Action and a hosted runner.
>
> I ran it on @selfactuated using an ARM64 machine and a microVM.

> That took the time down to 1m 26s!! About a 22x speed increase. https://t.co/zwF3j08vEV pic.twitter.com/ps21An7B9B
>
> — Alex Ellis (@alexellisuk) October 20, 2022

They limit maximum concurrency

On the free plan, you can only launch 20 hosted runners at once, this increases as you pay GitHub more money.

Builds on private repos are billed per minute

I think this is a fair arrangement. GitHub donates Azure VMs to open source users or any public repo for that matter, and if you want to build closed-source software, you can do so by renting VMs per hour.

There's a free allowance for free users, then Pro users like myself get a few more build minutes included. However, These are on the standard, 2 Core 7GB RAM machines.

What if you didn't have to pay per minute of build time?

What's wrong with self-hosted runners?

It's challenging to get all the packages right as per a hosted runner

I spent several days running and re-running builds to get all the software required on a self-hosted runner for the private repos for OpenFaaS Pro. Guess what?

I didn't want to touch that machine again afterwards, and even if I built up a list of apt packages, it'd be wrong in a few weeks. I then had a long period of tweaking the odd missing package and generating random container image names to prevent Docker and KinD from conflicting and causing side-effects.

What if we could get an image that had everything we needed and was always up to date, and we didn't have to maintain that?

Self-hosted runners cause weird bugs due to caching

If your job installs software like apt packages, the first run will be different from the second. The system is mutable, rather than immutable and the first problem I faced was things clashing like container names or KinD cluster names.

You get limited to one job per machine at a time

The default setup is for a self-hosted Actions Runner to only run one job at a time to avoid the issues I mentioned above.

What if you could schedule as many builds as made sense for the amount of RAM and core the host has?

Docker isn't isolated at all

If you install Docker, then the runner can take over that machine since Docker runs at root on the host. If you try user-namespaces, many things break in weird and frustrating aways like Kubernetes.

Container images and caches can cause conflicts between builds.

Kubernetes isn't a safe alternative

Adding a single large machine isn't a good option because of the dirty cache, weird stateful errors you can run into, and side-effects left over on the host.

So what do teams do?

They turn to a controller called Actions Runtime Controller (ARC).

ARC is non trivial to set up and requires you to create a GitHub App or PAT (please don't do that), then to provision, monitor, maintain and upgrade a bunch of infrastructure.

This controller starts a number of re-usable (not one-shot) Pods and has them register as a runner for your jobs. Unfortunately, they still need to use Docker or need to run Kubernetes which leads us to two awful options:

1. Sharing a Docker Socket (easy to become root on the host)
2. Running Docker In Docker (requires a privileged container, root on the host)

There is a third option which is to use a non-root container, but that means you can't use `sudo` in your builds. You've now crippled your CI.

What if you don't need to use Docker build/run, Kaniko or Kubernetes in CI at all? Well ARC may be a good solution for you, until the day you do need to ship a container image.

3. Can we fix it? Yes we can.

Actuated (*"cause (a machine or device) to operate."*) is a semi-managed solution that we're building at OpenFaaS Ltd.

A semi-managed solution, where you provide hosts and we do the rest

> A semi-managed solution, where you provide hosts and we do the rest.

You provide your own hosts to run jobs, we schedule to them and maintain a VM image with everything you need.

You install our GitHub App, then change `runs-on: ubuntu-latest` to `runs-on: actuated` or `runs-on: actuated-aarch64` for ARM.

Then, provision one or more VMs with nested virtualisation enabled on GCP, DigitalOcean or Azure, or a bare-metal host, and install our agent. That's it.

If you need ARM support for your project, the a1.metal from AWS is ideal with 16 cores and 32GB RAM, or an Ampere Altra machine like the c3.large.arm64 from Equinix Metal with 80 Cores and 256GB RAM if you really need to push things. The 2020 M1 Mac Mini also works well with Asahi Linux, and can be maxed out at 16GB RAM / 8 Cores. I even tried Frederic's Parca job on my Raspberry Pi and it was 26m30s quicker than a hosted runner!

Whenever a build is triggered by a repo in your organisation, the control plane will schedule a microVM on one of your own servers, then GitHub takes over from there. When the GitHub runner exits, we forcibly delete the VM.

You get:

- A fresh, isolated VM for every build, no re-use at all
- A fast boot time of ~ <1-2s
- An immutable image, which is updated regularly and built with automation
- Docker preinstalled and running at boot-up
- Efficient scheduling and packing of builds to your fleet of hosts

It's capable of running Docker and Kubernetes (KinD, kubeadm, K3s) with full isolation. You'll find some examples in the docs, but anything that works on a hosted runner we expect to work with actuated also.

Here's what it looks like:

Want the deeply technical information and comparisons? Check out the FAQ

You may also be interested in a debug experience that we're building for GitHub Actions. It can be used to launch a shell session over SSH with hosted and self-hosted runners: Debug GitHub Actions with SSH and launch a cloud shell

Wrapping up

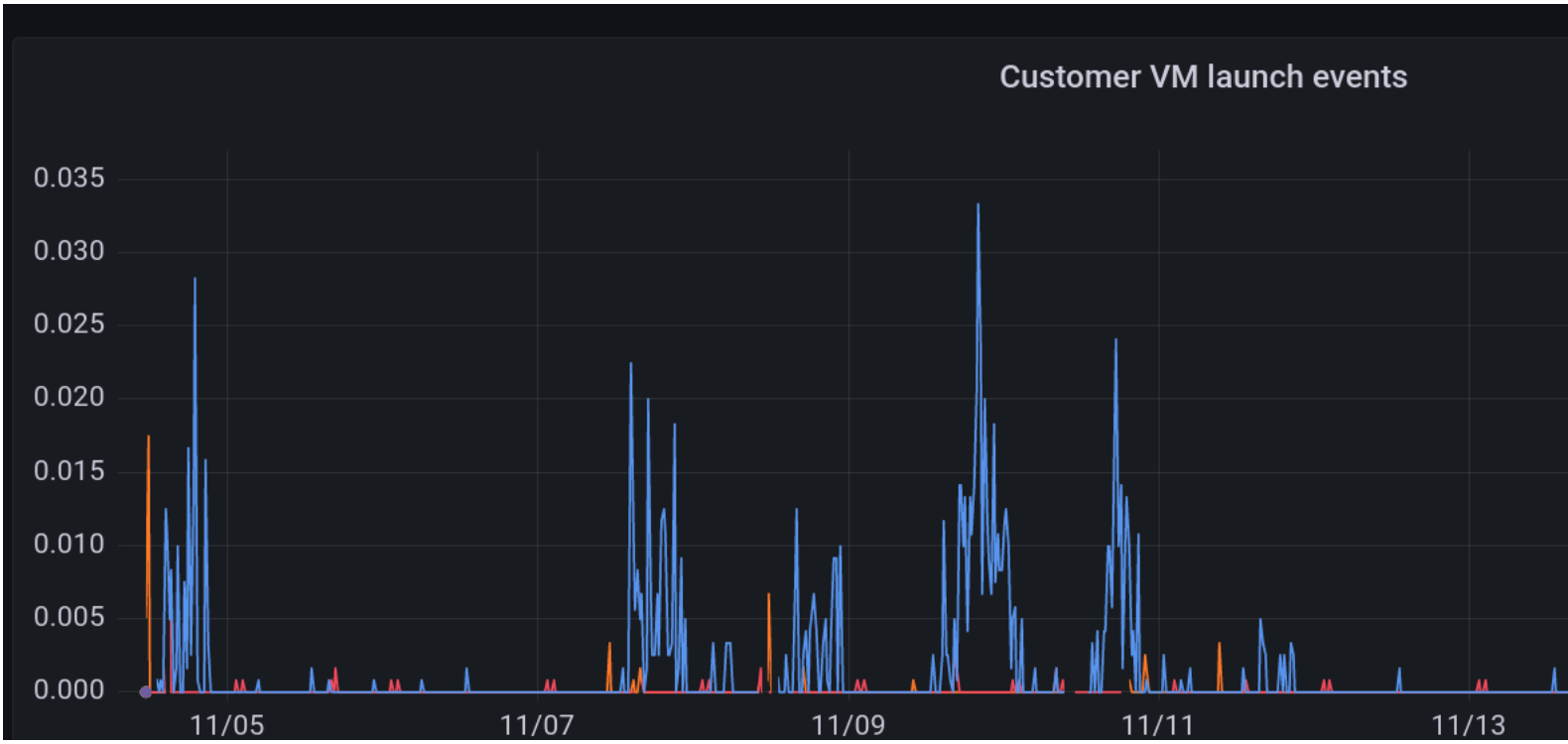
We're piloting actuated with customers today. If you're interested in faster, more isolated CI without compromising on security, we would like to hear from you.

Register for the pilot

We're looking for customers to participate in our pilot.

Register for the pilot

Actuated is live in pilot and we've already run thousands of VMs for our customers, but we're only just getting started here.



Pictured: VM launch events over the past several days

Other links:

- Read the FAQ
- Watch a short video demo
- Follow actuated on Twitter

What about GitLab?

We're focusing on GitHub Actions users for the pilot, but have a prototype for GitLab. If you'd like to know more, reach out using the Apply for the pilot form.

Just want to play with Firecracker or learn more about microVMs vs legacy VMs and containers?

- Watch A cracking time: Exploring Firecracker & MicroVMs
- Try my firecracker lab on GitHub - alexellis/firecracker-init-lab

What are people saying about actuated?

"We've been piloting Actuated recently. It only took 30s create 5x isolated VMs, run the jobs and tear them down again inside our on-prem environment (no Docker socket mounting shenanigans)! Pretty impressive stuff."

Addison van den Hoeven - DevOps Lead, Riskfuel

"Actuated looks super cool, interested to see where you take it!"

Guillermo Rauch, CEO Vercel

"This is great, perfect for jobs that take forever on normal GitHub runners. I love what Alex is doing here."

Richard Case, Principal Engineer, SUSE

"Thank you. I think actuated is amazing."

Alan Sill, NSF Cloud and Autonomic Computing (CAC) Industry-University Cooperative Research Center

"Nice work, security aspects alone with shared/stale envs on self-hosted runners."

Matt Johnson, Palo Alto Networks

"Is there a way to pay github for runners that suck less?"

Darren Shepherd, Acorn Labs

"Excited to try out actuated! We use custom actions runners and I think there's something here "

Nick Gerace, System Initiative

It is awesome to see the work of Alex Ellis with Firecracker VMs. They are provisioning and running Github Actions in isolated VMs in seconds (vs minutes)."

Rinat Abdullin, ML & Innovation at Trustbit

"This is awesome!" (After reducing Parca build time from 33.5 minutes to 1 minute 26s)

Frederic Branczyk, Co-founder, Polar Signals