changelog.com/posts

The big idea around unikernels

by **Ian Eyberg** • 2021-12-01 • #cloud +1

Did you know Linux is approaching 30 years old? Did you know Unix is around 50 years old? I'm not knocking them — I've been using Linux since they were distributed on floppies. However, Unix was built for machines like the PDP-7 — you know, the kind that takes up an entire wall.

The PDP-7 - a minicomputer produced by Digital Equipment Corporation

But, have you ever wondered what an operating system built in 2022 would look like?

Today, cloud infrastructure is so complex that devops/SRE folks are attracting salaries that sometimes exceed normal software engineers.

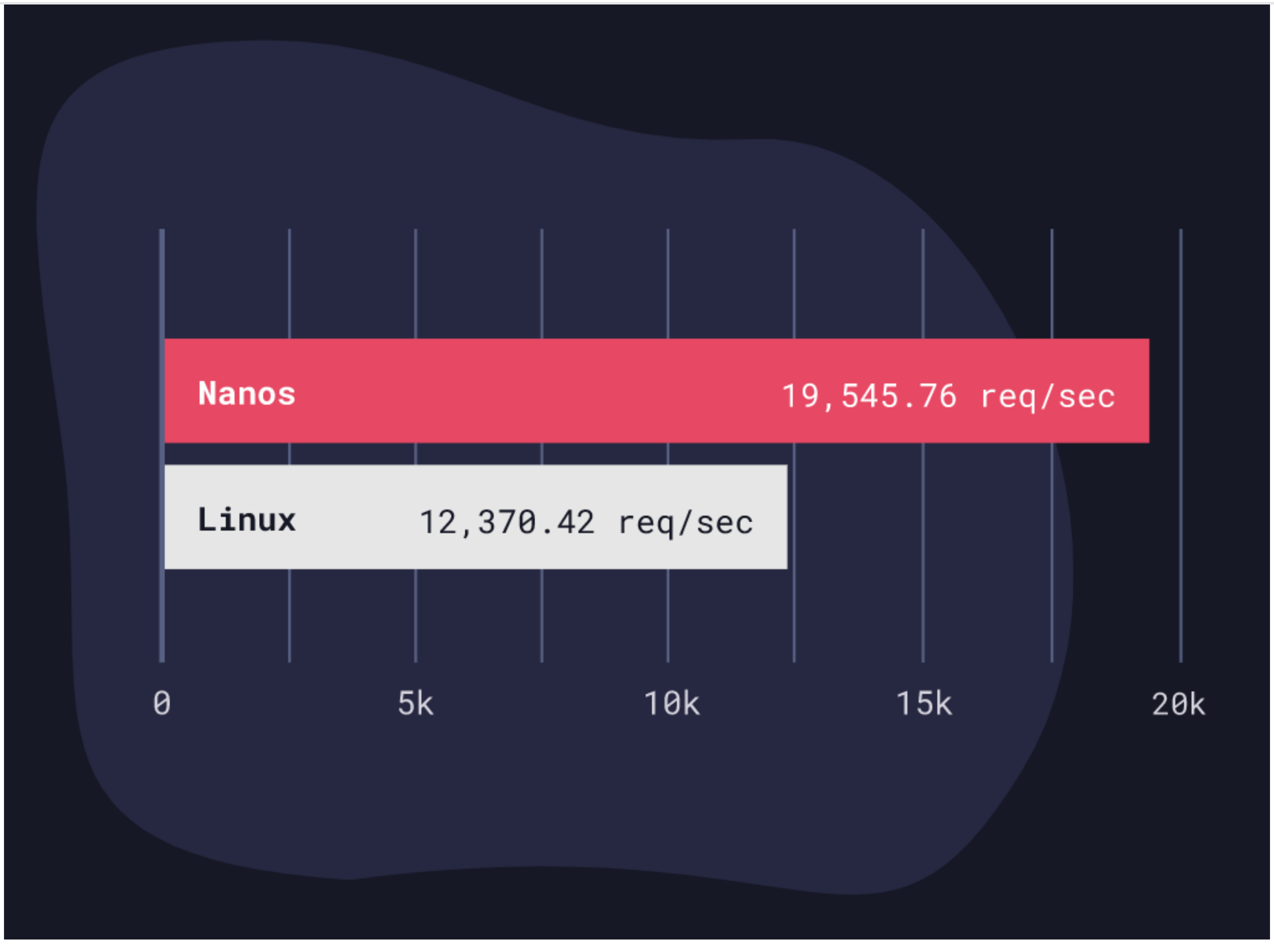
Cybersecurity is so insanely bad that there is at least one major data breach *every single week* now. If you look at illicit crypto miner bots that infect k8s clusters, they literally have code that hunts down other bots and kicks them off the server before doing anything else. Bots having a turf war - that's the state of our cybersecurity.

Honestly, it is good that Hollywood is not real life because all the cybersecurity woes are not even close to how bad things could really be. It isn't like someone has hacked a nuclear power plant before — **cough**.

Unikernels can be fast

We run websites like ops.city and nanos.org as Go unikernels on Google Cloud. We routinely benchmark webservers running 2X faster on Google and 3X faster on AWS. We've clocked things like Redis pushing 20% on average more on its benchmark tool.

The numbers don't lie.



Go Webserver on Nanos vs Debian 10 "Buster" Linux on GCP G1-Small

There's no special algorithm being used here - it's just the architecture that allows this to happen. Some people ask if you can just use something like Alpine and get the same results. Unfortunately, the answer is no. Linux is like 30M lines of code and not everything can just be excluded by building a custom kernel.

For instance, when you remove things like multiple process support you start to see how infectious that support truly is. That touches the scheduler, that touches shared memory, that touches signaling, that touches *a lot* of stuff, so it's not something where you can just go in and ifdef/patch it out. Similarly, seccomp doesn't do everything we want either.

Unikernels can be fast to boot, too. This is a weird point to make, though, as things like Firecracker usually come to people's minds when it's mentioned. However, unikernels expose other interesting quirks that you'd never see otherwise when you deploy them. For instance, we crafted a small little Rust webserver that sat on Google and injected a crash on each request. It would immediately reboot, ready to serve the next request instantly. However, every time it rebooted it came with a brand new memory layout cause of ASLR which from a security perspective is very interesting. That's one way to seriously screw with an attacker's head!

Honestly, we haven't even done the cool stuff with unikernels yet. Imagine being able to tune your unikernel depending on whether it's network heavy or filesystem heavy. Imagine how much easier binary weaving and automated dead code removal can be in such a system. Some applications like to link to every library under the sun even if they only use one function from it.

Unikernels give us an excellent opportunity to do some *long-overdue* house cleaning.

Unikernels are isolated

The isolation of unikernels is what attracted me to them, coming from the security industry. No users or passwords - interesting. No shell. No remote login or ssh. More importantly, unikernels can only run one and only one application per VM. So that means in your typical webserver stack, you have one VM as the actual webserver and another for your database (you probably already do this anyway).

This concept goes much deeper though. When you think about an attacker breaking into your servers, it is precisely like a robber breaking into your house. They come through by kicking in the door or smashing a window but that's not *why* they are breaking into the house. They come for the guns, jewels, money, and flatscreen TVs. For hackers, it is the same. Breaking your software by exploiting a bug is just the way into the server. That is why patch management and vulnerability scanning doesn't actually reduce the number of security breaches in the news.

The end goal for the attacker is to run *their* programs - they couldn't care less about yours. Those programs could be running `mysqldump` against your database, or maybe installing a crypto miner. Regardless, it's always other pieces of software - usually *many* pieces of software. That's why most exploits focus their shellcode on forking a shell. A shell is inherently built to run multiple programs. When you type `ls`, `ps aux`, or `cd` you call those commands, but they are other programs. None of that works in unikernel land.

So, while you might be able to obtain the use of some memory on a vulnerable piece of software you probably now have a tough time doing anything useful with it. Has anyone seen a MySQL client written in pure ROP gadgets?

Unikernels are simple

The past decade has seen massive infrastructure creep and while it can be easy to blame the tooling, companies are publishing massive amounts of software now.

So how simple? You can push your application out to Google Cloud with just two commands. This same set of commands only takes ~20 seconds before your site is live on AWS:

```
ops image create -c config-prod-myapp.json myapp -i myapp -t gc
ops instance create myapp -c config-prod-myapp.json -z us-west2-
```

When first introduced to unikernels, many people wonder, “Where is the Kubernetes for orchestrating unikernels? Do I have to invest all this time and energy in learning yet another thing?” The answer is it doesn’t exist, and it doesn’t *need* to exist. Why? When we deploy unikernels to AWS or Google, we create a new AMI every time you hit the deploy button (don’t worry, the entire process can take less than 20 seconds). The VM doesn’t have Linux installed on it at all - it’s just your app. The underlying storage device and networking are all taken care of by your cloud of choice, so while you can configure it, you don’t have to manage it. This is a significant distinction. You can think of it as serverless without the lock-in because the same commands work on any cloud provider out there. By shifting this burden of responsibility to the cloud provider, you force the cloud to do what it does best (manage infrastructure), and you do what you do best (manage your application).

This makes pushing your application to prod painfully easy.

If the application crashes (which happens to every software developer out there, no matter how good they are), the entire VM crashes. What’s interesting about debugging this is that it actually is much easier to debug than a Linux system. Why? Because there is only one program in question. You aren’t whipping out `lsof` to figure out what process is spewing out crap connections or which process didn’t have a proper log rotation setup and prevented you from SSHing into the instance.

In fact, we used to crash the database backing our Radar monitoring service quite a lot earlier on. Eating our own dogfood and running it as a Nanos unikernel allowed us to enumerate many issues. Networking bugs, storage bugs, the whole works. That’s why we know it works well now because we’ve had to go in and fix a ton of bugs. However, when could attach `gdb` to it and pinpoint directly where it was having an issue. We could export the stack trace as well. You can actually clone these VMs in prod, in real-time with no downtime, and monkey around with the clones too. I can imagine all sorts of interesting things you can do besides debugging with functionality like that.

I remember another time we were debugging a networking issue on Google. Someone had reported our site was not showing up for them, but it was showing up for us. Strange. It turns out the MTU was set to a different number. We were able to pinpoint the issue easily and fix the TCP/IP stack.

Unikernels are so much easier to debug than normal Linux systems.

Virtualization, SMP, and the second tech boom

The old paradigm of one server with one operating system hosting many applications doesn't make sense anymore.

If you work at a company that is more than 10 or 20 people, you don't have one server. You have many servers. If you work at a company like Uber or Airbnb, you don't have one database. You have *thousands* of databases. That's thousands of operating systems that have to be managed as well.

Do you remember this graphic?

Google wrote a paper, then wrote another paper, and so many papers later ended up writing a book that basically said **the datacenter is a warehouse-sized computer**.

The big idea around unikernels, at least when it comes to the cloud, is that if the datacenter is the computer, then the cloud is its operating system — so let's start treating it like one and stop micro-managing thousands of individual ones.

Ready to run your first unikernel? Check out **ops.city** and **nanos.org**. There is no better way of understanding what these are and how they work than to try it out.

you@example.com Subscribe

Discussion

Sign in or **Join** to comment or subscribe

BANDWIDTH

CLOUD HOSTING

FEATURE FLAGS

OBSERVABILITY