FLOX

When Everything Looks Like a Container

Michael Stahnke – VP of Engineering and Growth

flox.dev

@stahnma

Rethinking 15 Years of Cloud-Native Defaults

CATERPILLAR®

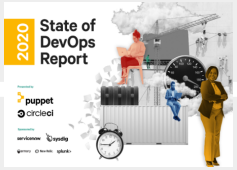
puppet

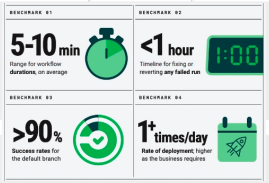
circleci

FLOX

EPEL

Uplevel





Ruby

AIX

solaris

SUSE

CISCO

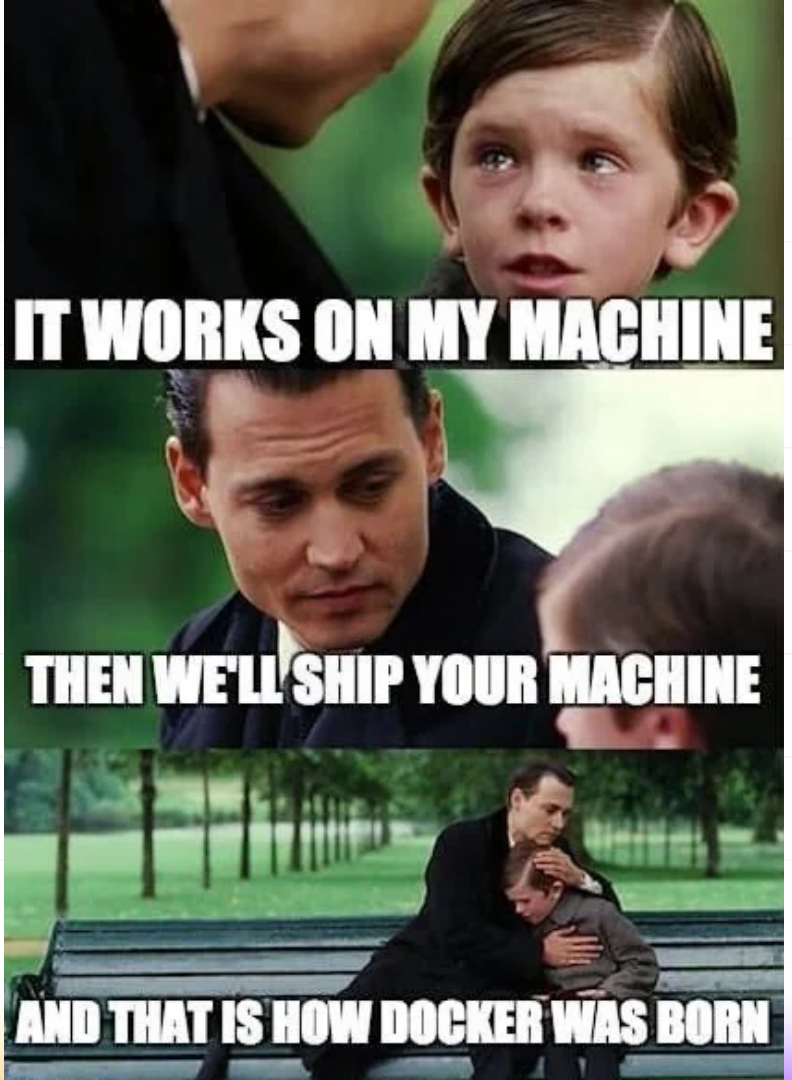
cumulus

ARISTA

vmware

Containers solved one problem.

Developers didn't want to package software properly.


IT WORKS ON MY MACHINE
THEN WE'LL SHIP YOUR MACHINE
AND THAT IS HOW DOCKER WAS BORN

This is NOT an anti-container talk.

Containers solved real problems.

Consistency.
Portability.
Deployment confidence.

Docker (2013) was genuinely revolutionary.

It made shipping software dramatically easier.

Give containers their flowers.

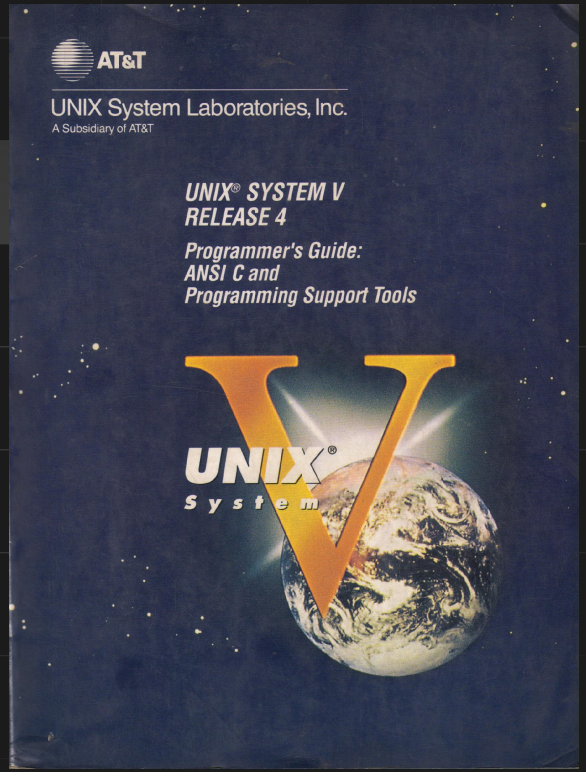
FLOX

How We Forgot What an OS Can Do

flox.dev

@stahnma

Unix (80s–2000s): multi-user, multi-process, shared resources



Multiple users.
One machine.

The OS handled isolation: users, groups, permissions, quotas, communication

You could wall a message to everyone logged in.

You could talk to another user in real time.

The machine was a shared, living system.

VMs: one app per virtual machine

Threw away multi-user.
Kept the OS.

Containers:
one process per container

Threw away multi-user.
Threw away most of the OS.

90% of the OS is dead weight.

A younger sysadmin who'd never seen wall or talk.

Never seen two people use the same computer at the same time.

We didn't solve isolation.

We forgot a multi-user experience was built in from the ground up

Developer onboarding?
Container.

CI pipeline?
Container.

Local dev environment?
Container.

Edge deployment?
Container.

ML workflow?
Container.

When you have a hammer...

...but this hammer came with an entire hardware store.

I have 3 HTML files and a CSS stylesheet.

OK first you need a Dockerfile

Put Nginx in a container

And a registry to store the image

And an orchestrator to run it

And a service mesh so it can talk to things

And an ingress controller

And a secrets manager

And a CI/CD pipeline to build the container

And a scanning tool

And a policy engine

And observability tooling to watch all of this

And a team to maintain the platform that runs all of this

...it's 3 HTML files and a CSS stylesheet.

I have a function that runs once a day.

Ship it as a 500MB container image to Lambda.

A 5MB zip
would have done the job.

Amazon makes more money when you ship 500MB

The incentives are not aligned.

The platform profits from your inefficiency.

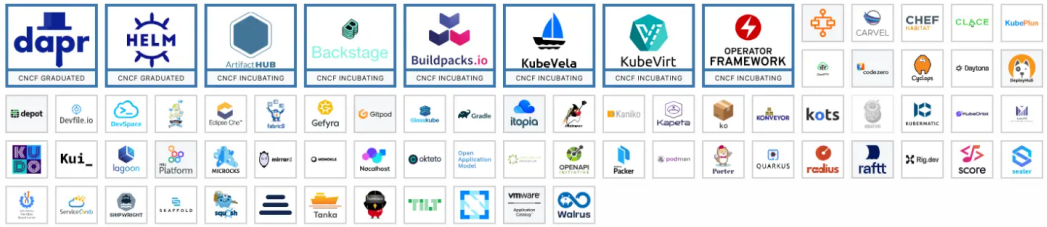
FLOX

The Complexity Tax

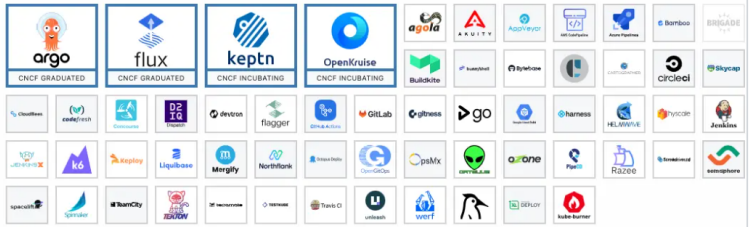
flox.dev

@stahnma

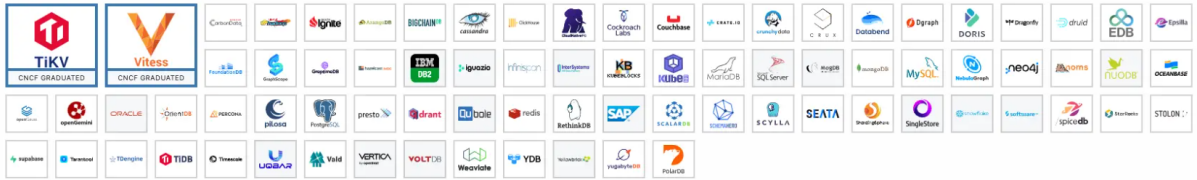
Application Definition & Image Build



Continuous Integration & Delivery



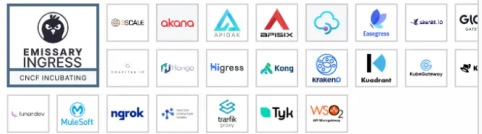
Database



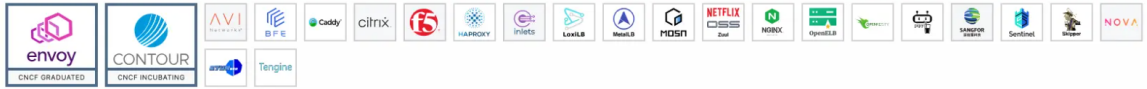
Streaming & Messaging

Scheduling & Orchestration

API Gateway



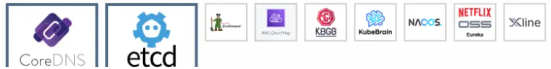
Service Proxy



Remote Procedure Call

Service Mesh

Coordination & Service Discovery



No single human can understand all of this.

Every problem → new abstraction

Every abstraction → complexity tax

We didn’t stop to ask: is this layer necessary?

FLOX

We rebuilt the OS, but worse

flox.dev

@stahnma

Process isolation → Containers + orchestration

Process supervision (systemd) → K8s controllers + health checks

IPC (pipes, sockets) → Service meshes + gRPC

Logging (syslog) → ELK / Fluentd / Loki

ps, top, strace → Prometheus + Grafana + Jaeger + OpenTelemetry

User permissions → RBAC + OPA + admission controllers

Filesystem permissions → Vault + sealed secrets

Cron → Kubernetes CronJobs

talk, wall, signals → Webhooks + event buses

/etc/hosts, DNS → Service discovery + ingress controllers

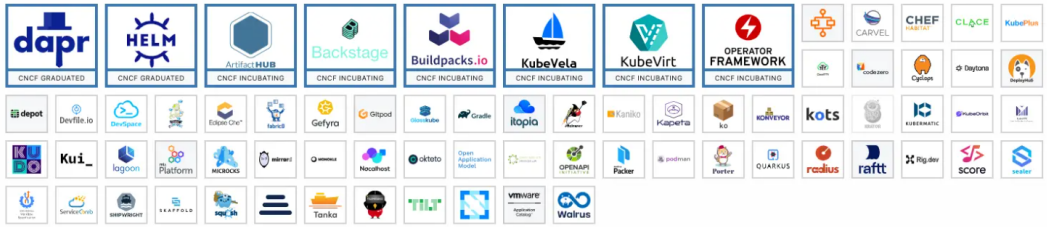
Unix	Cloud Native
Process Isolation	Containers & Orchestration
Process Supervision (systemd)	K8s Controllers & Health Checks
IPC (pipes, sockets)	Services Meshes and gRPC
Logging (syslog)	ELK / Fluent / Loki
ps, top, strace	Prometheus & Grafana & Jaeger & OpenTelemetry
User permissions	RBAC & OPA & admission controllers
Filesystem permissions	Vault & sealed secrets
Cron	Kubernetes CronJobs
talk, wall, signals	Webhooks & event buses
/etc/hosts, DNS	Service discovery & ingress controllers

We didn't add capabilities.

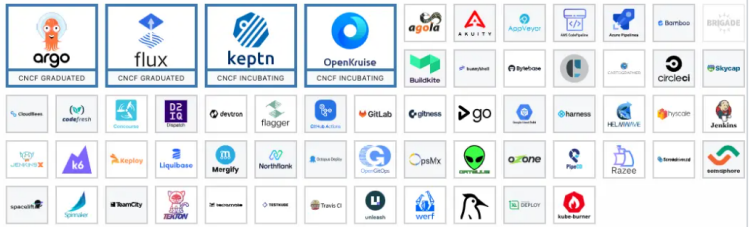
We decomposed a working system

and reassembled it from
a thousand vendor-sponsored pieces.

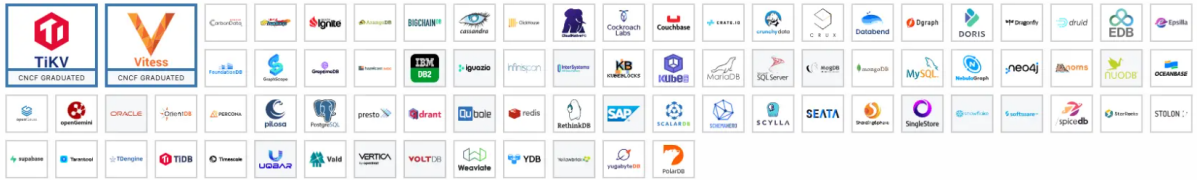
Application Definition & Image Build



Continuous Integration & Delivery



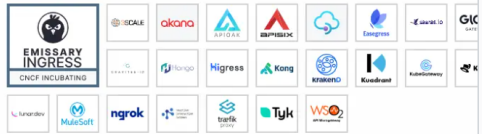
Database



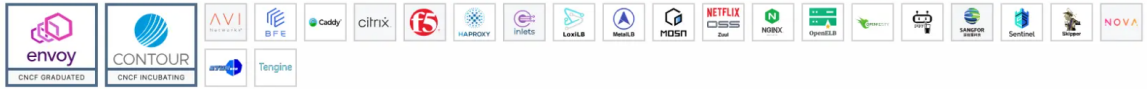
Streaming & Messaging

Scheduling & Orchestration

API Gateway



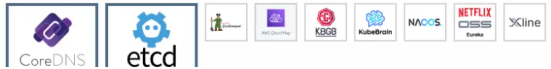
Service Proxy



Remote Procedure Call

Service Mesh

Coordination & Service Discovery



FLOX

Containers as Technical Debt Laundering or Default Swap Derivatives

flox.dev

@stahnma

Messy app?
Wrap it in a container.

The debt didn't go away.

You put it in a box with a label on it.

The infrastructure equivalent of shoving everything in a closet before guests arrive.

Your house looks clean.

Open the closet and it all falls out.

It works in the container = I stopped investigating why it's broken

We didn't fix our dependency mess.

We baked it into a file format and called it an image.

FLOX

Chronicles:
The Six-Service Outage

Production outage.

Errors showing up in Service A.

The problem was 6–8 services upstream.

An expired cache in a service nobody was watching.

Debugging took heroics.

On a simpler system: strace → "this file is missing." Done.

Complexity doesn't just make building harder.

It makes debugging dramatically harder.

And you debug more than you build.

The cost of developing software rounds to zero when compared to operating it in perpetuity

FLOX

The Distributed Monolith

flox.dev

@stahnma

All the operational complexity of distributed systems.

All the coupling of a monolith.

Services that can't deploy independently.

Services that can't deploy independently. (At least not all the time)

Synchronous call chains 8 services deep.

A well-structured monolith: one strace away from the answer.

We split the monolith into microservices

and got a distributed monolith.

Somehow that's worse.

It might not be good, but at least it’s expensive.

FLOX

The Sidecar Explosion

flox.dev

@stahnma

You deployed 1 thing.

You're running 4.

Application container

Logging sidecar

Service mesh proxy

Security sidecar

From run my app to orchestrate four containers so my app can write a log line

FLOX

The Monitoring Paradox

flox.dev

@stahnma

More complex system → more monitoring needed

More monitoring → more complex system

Some orgs spend more compute on observability than on the actual workload.

ps was free.
top was free.
strace was free.

You now need monitoring for your monitoring.

We optimized for dev.

We forgot about debug.

We made it trivially easy to ship something nobody can troubleshoot.

FLOX

Container Golf

flox.dev

@stahnma

Alpine! No, distroless!
No, scratch!

A practice that only exists because of the container model.

Scanning the image, not the running container.

latest today ≠ latest in two weeks

Mutable tags are one area I won’t be gentle on. I think it might be the single biggest mistake in the container ecosystem.

apt-get update gives you
different results
on different days.

A packaging format with the same reproducibility problems it was supposed to solve.

And we're about to do it all again with AI.

GPU model-serving containers are 15GB+

Same golf.
Same theater.
10–30x the size.

FLOX

Frozen Entropy

flox.dev

@stahnma

Python 3.11. Node 18.
Everything works. Ship it.

Months pass. Years pass.
It still "works."

A system at rest will remain online. — Newton (One of Newton's lesser known laws)

Unpatched.
Accumulating CVEs.

Containers let you freeze entropy.

Sounds like a feature.

Until you've also frozen your security posture.

Containers didn't make us lazy.

They made it easy to stop moving forward and call it stability.

The forcing function of a hardware migration was lost

FLOX

And there’s just too much yaml

flox.dev

@stahnma

We configure the most complex distributed systems on earth

in a whitespace-significant language.

The Norway Problem:
NO → false

Helm: YAML templated with Go templates generating more YAML

Programming, but worse.
No type system.
No compiler. No debugger.

FLOX

The Uncomfortable Truths

FLOX

You Don't Have a Scale Problem

flox.dev

@stahnma

K8s was designed for Google's scale.

You are not Google.

You may have an adoption problem.

You may have a product-market fit problem.

You may have a unit economics problem.

Scaling is a late-stage problem.

If you have it, congratulations.

Premature scaling is premature optimization.

Premature optimization is the root of all evil. — Donald Knuth

FLOX

The Multi-Cloud Portability Myth

flox.dev

@stahnma

We'll use K8s
so we can run anywhere.

Almost nobody actually does this.

Your Terraform is cloud-specific.
Your monitoring is cloud-specific.
Your billing is cloud-specific.

They traded vendor lock-in for ecosystem lock-in.

We added K8s for portability

And now need a platform team to manage the portability layer we've never used to be portable.

FLOX

Resume-Driven Development

flox.dev

@stahnma

I need K8s experience to get my next job.

That's a real incentive.
And it's not evil.

But it should be named.

Once K8s is in,
nobody wants to remove it.

If your career rewards complexity, who's incentivized to simplify?

FLOX

"Cloud-Native" as Gatekeeping

flox.dev

@stahnma

Not containers + K8s?
Not "cloud-native."

Not cloud-native?
Not "modern."

Not modern?
Can't attract talent.

Running VMs?
That's legacy.

Technology is the only industry where “legacy” is a bad word.

But, “legacy” means it makes money.

Using a monolith?
You'll never scale.

Questioning the stack feels like questioning whether you belong.

Judge by outcomes, not by labels.

FLOX

Get Good

flox.dev

@stahnma

K8s isn't that hard, get good.

If you've done this for 10 years, it IS second nature.

Your expertise is not transferable to every new hire.

"It's easy for me"
is survivorship bias.

The question isn't "can an expert operate this?"

It's "should this require an expert?"

Technology-oriented vs. outcome-oriented.

The best infrastructure disappears and gets out of your way.

FLOX

What Could Be Different

FLOX

Network Costs

flox.dev

@stahnma

Containers at the edge: full images over constrained networks.

Change 1 of 30 packages → ship 15 new layers (Docker)

Change 1 of 30 packages → ship 1 delta (Nix/Flox)

When bandwidth is expensive, this isn't academic. It's budget.

Do you need a container runtime on a device with 512MB of RAM?

FLOX

Know Your Options

flox.dev

@stahnma

Bare metal / VMs with good config management

Systemd units.
Yes, really.

Single binaries (Go, Rust).
No (minimal) runtime dependency.

Nix closures / Flox environments

The complete dependency graph, computed and hermetic.

Not "ship the whole OS."
Ship exactly what you need.

FLOX

Firecracker / MicroVMs

flox.dev

@stahnma

AWS built Firecracker because containers weren't the right primitive.

Boots in ~125ms.
Minimal footprint.

When the cloud providers need isolation, they don't reach for containers.

FLOX

WebAssembly (WASM/WASI)

flox.dev

@stahnma

Solomon Hykes, Docker co-founder:

"If WASM+WASI existed in 2008,
we wouldn't have needed
to create Docker."

WASM modules are kilobytes.
They start in microseconds.

Portable.
Sandboxed by default.

It's early. But for a huge category of workloads: dramatically simpler.

Containers + K8s —
when you genuinely need it.

The menu is longer than most people realize.

Multi-user Linux is still a thing.

The OS already knew how to do this.

FLOX

The Decision Framework

flox.dev

@stahnma

1. What problem am I actually solving?

1. What problem am I actually solving?
2. Does the container solve it, or just defer it?

1. What problem am I actually solving?
2. Does the container solve it, or just defer it?
3. What's the complexity tax, and who pays it?

1. What problem am I actually solving?
2. Does the container solve it, or just defer it?
3. What's the complexity tax, and who pays it?
4. Could a simpler tool solve it with fewer layers?

1. What problem am I actually solving?
2. Does the container solve it, or just defer it?
3. What's the complexity tax, and who pays it?
4. Could a simpler tool solve it with fewer layers?
5. Am I choosing this because it's right, or because it's default?

FLOX

AI: We're About to Make Every Mistake Again

flox.dev

@stahnma

How do I serve a model?
→ Put it in a container.

Images are 15GB+.

Same complexity accretion.
10x the speed.

But AI might break the cycle.

What if AI can actually solve the packaging problem?

Instead of
put your laptop in production

Let the tooling figure out what your app actually needs.

Will we learn from 15 years of container defaults?

Or sleepwalk into the same pattern with AI?

FLOX

The Close

You have permission to question the defaults.

You are up against the most powerful force in the universe.

Organizational Inertia.

"This is just how we do it"
is not an engineering argument.

It's not about blame.

With what we know now, would we make the same choice?

Frame it as simplification, not regression.

15 years of building an incredible ecosystem. That matters.

The mark of maturity isn’t adopting every tool.

It's knowing when to put one down.

The goal was never containers.

The goal was shipping software
that works, reliably,
to the people who need it.

If a container gets you there: use it.

If something simpler gets you there: use that.

Stop asking
how do I containerize this?

Start asking
what does this actually need?

Be outcome-oriented.
Solve the problem in front of you.

Resist the pull of the default.

flox.dev