Who Knew Dogfood Could Taste This Good?

A WebAssembly In Production Story

Brooks Townsend and Taylor Thomas, Cosmonic

Who are we?

Brooks Townsend

- Lead Software Engineer at Cosmonic
- wasmCloud maintainer
- Serial open source contributor
- Brewer of elixir, Wasm enjoyer
- Demo enthusiast

Taylor Thomas

- Director of Customer Engineering at Cosmonic
- Rustacean
- Co-creator of Krustlet and Bindle
- Open Source Maintainer
- Emeritus Helm Maintainer

Agenda

- What is WebAssembly?
- What’s this wasmCloud and Cosmonic thingy?
- Architecture
- Deep Dive
- Lessons Learned
- What happens next?

Neither Web, nor Assembly

Open W3C Standard

Open and widely supported standard

Safe & Secure

Deny by default secure sandbox, featuring capability driven permissions

Efficient and fast

Small size and near-native execution speed

Polyglot

Choice of deployment language means ability to reuse existing libraries

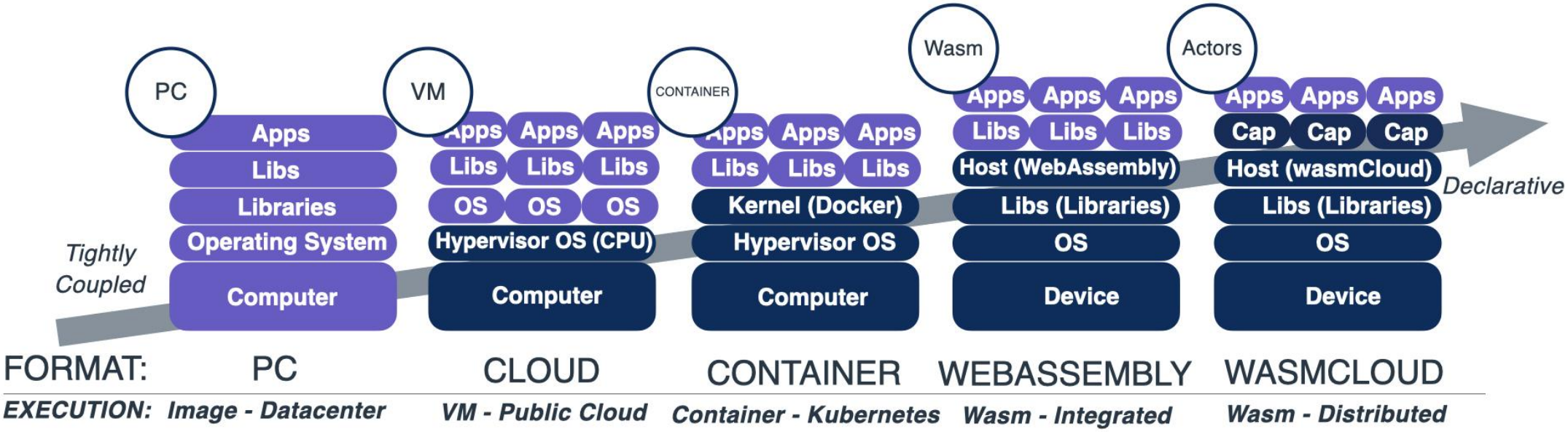
Portable

WebAssembly runs in all major browsers

But there are some gaps

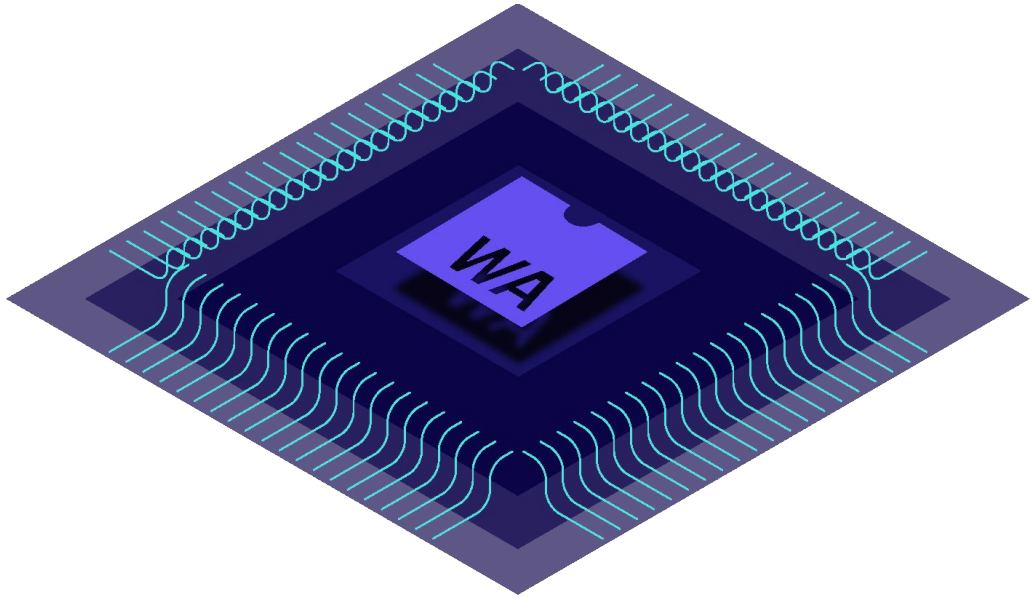
- Language support is still limited (but quickly growing!)
- Networking has come along, but is still rough around the edges
- Still have to compile your dependencies into the final binary (working on this too!)
- Numbers in, numbers out

Modern Computing Env



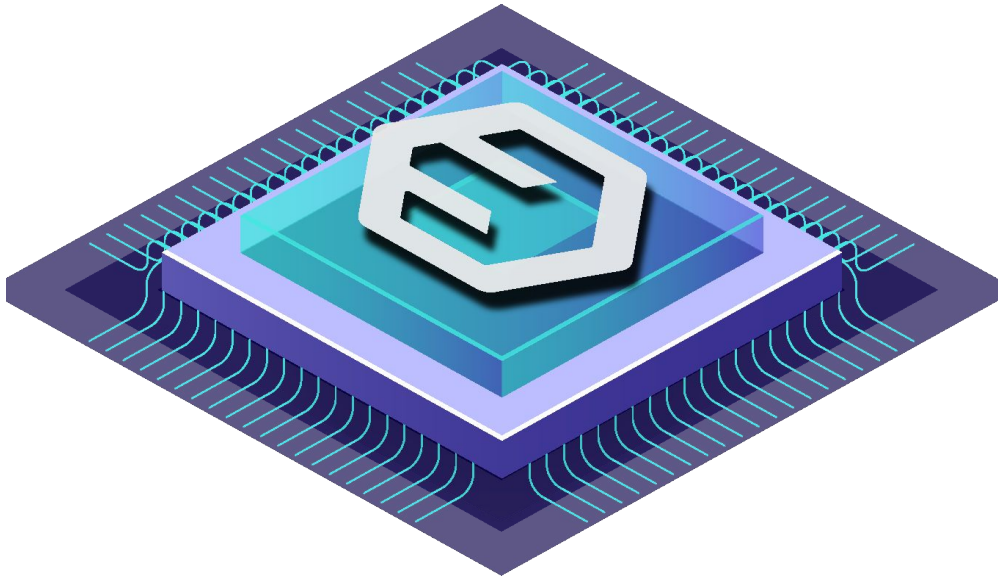
Legend: Developer Provided / Device / Computer

WebAssembly Host Runtime



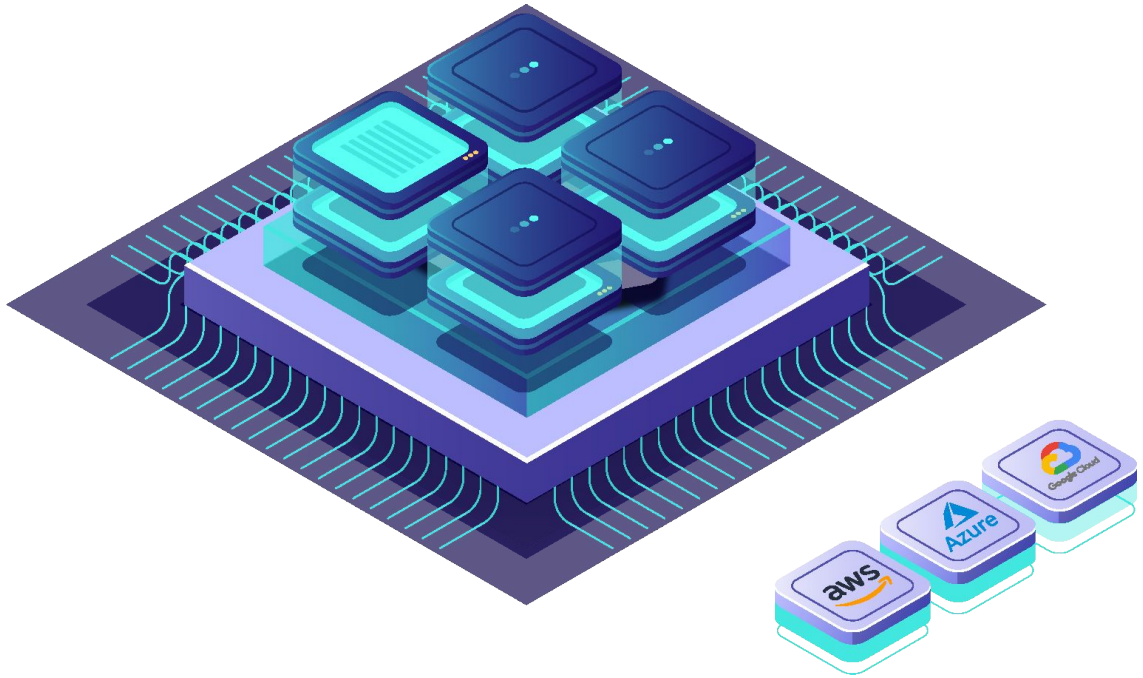
- Portable
- Secure
- Small
- Fast
- Language agnostic

wasmCloud Application Runtime



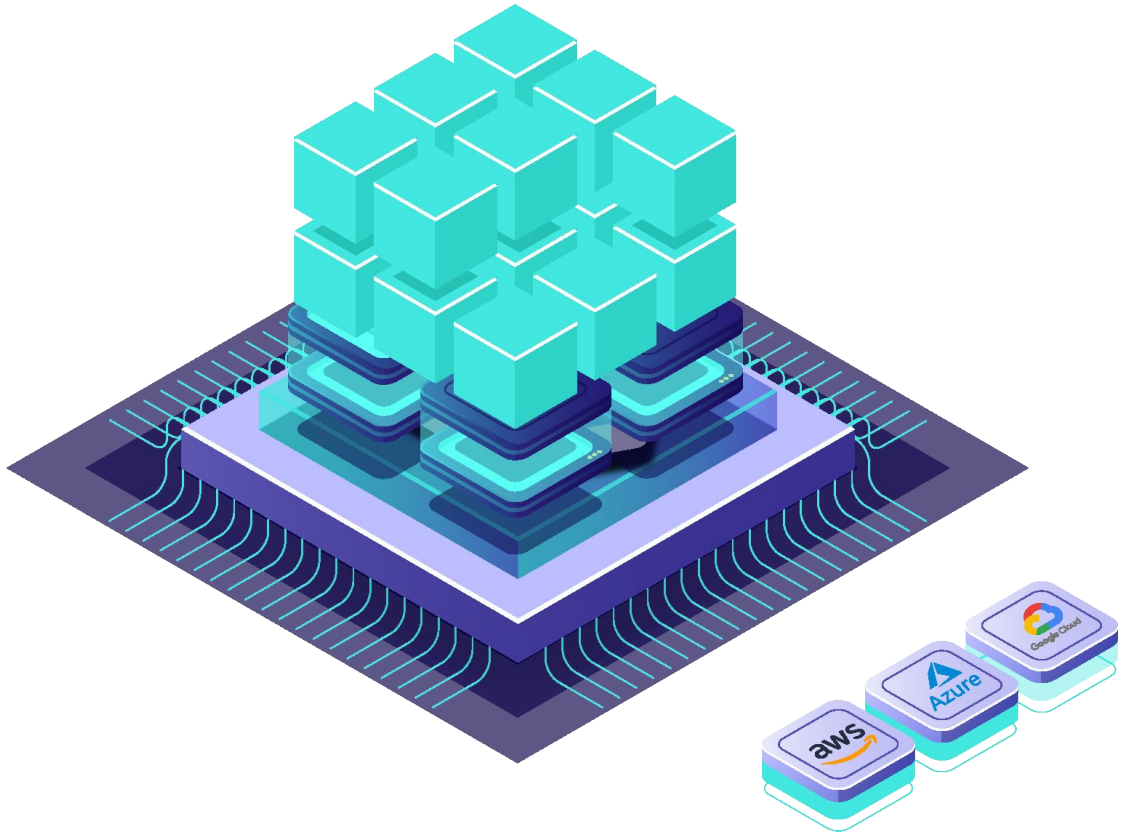
- Removed boilerplate code
- Secure access to capabilities
- Elixir/OTP - Extreme Scalability
- Horizontally and vertically scalable, stateless actors

Capabilities



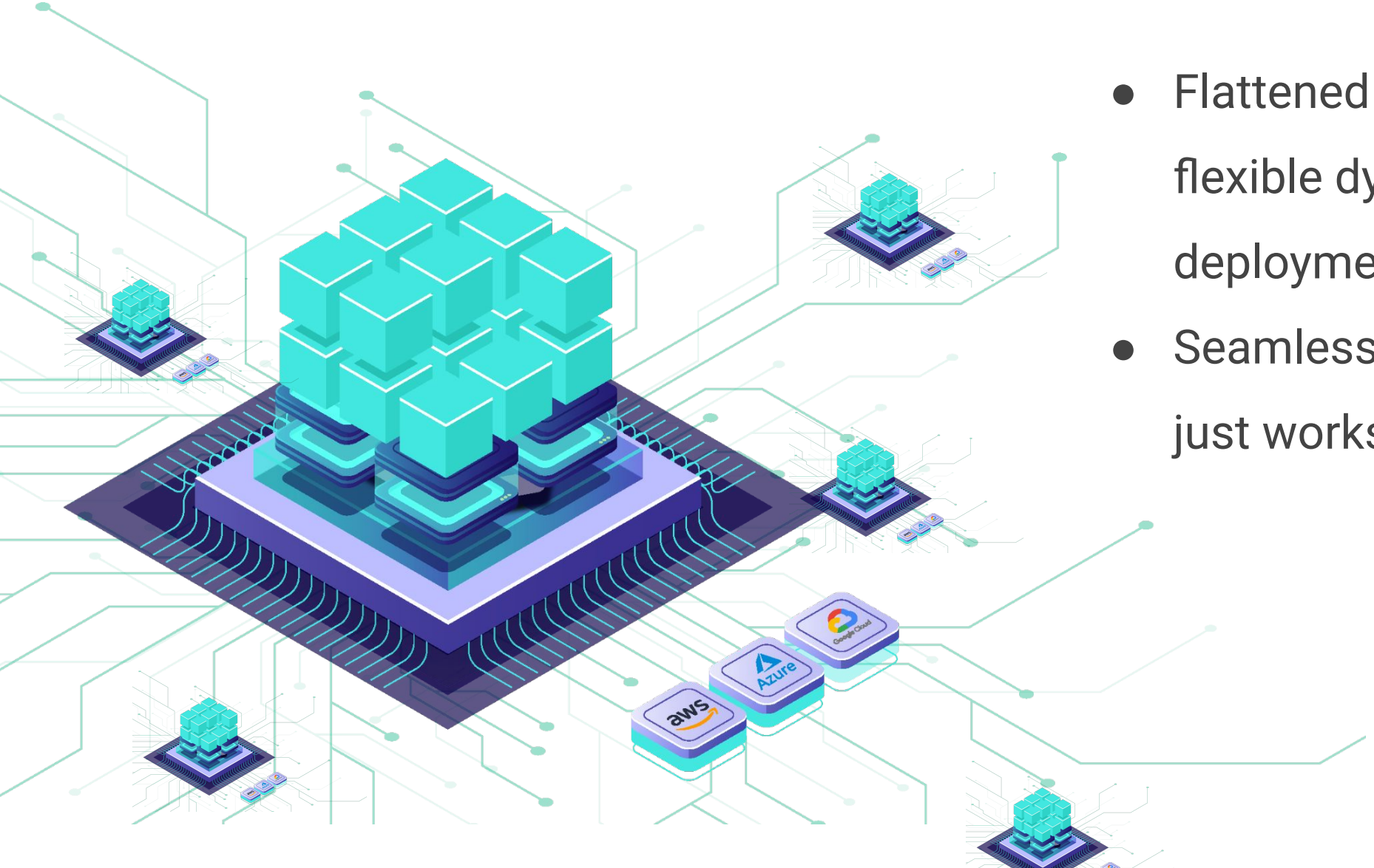
- **Maintain & update centrally - stop distributing vulnerable boilerplate**
- Runtime choice of capabilities, hot swap
- Contract driven design

Composable Actors



- Implement your business logic
- Stateless and reactive
- Easy to develop & low boilerplate
- Tiny footprint, portable & scalable

Lattice Network



- Flattened topology, enables flexible dynamic deployments
- Seamlessly connected, “it just works”

What is Cosmonic?

- Making it a painless experience to develop server-side wasm apps
- Easy cross-platform/cloud/device management
- Someone has to manage the thing at some point
- Everything is built on wasmCloud

The Architecture


made with
unscreen

Dogfooding 101

- When we started the product, we decided that we wanted to be Customer 0
- We were making some strong claims, so we wanted to put those to the test
- The learning served a dual purpose:
  - It improved our open core
  - Which in turn improved our customer’s experience

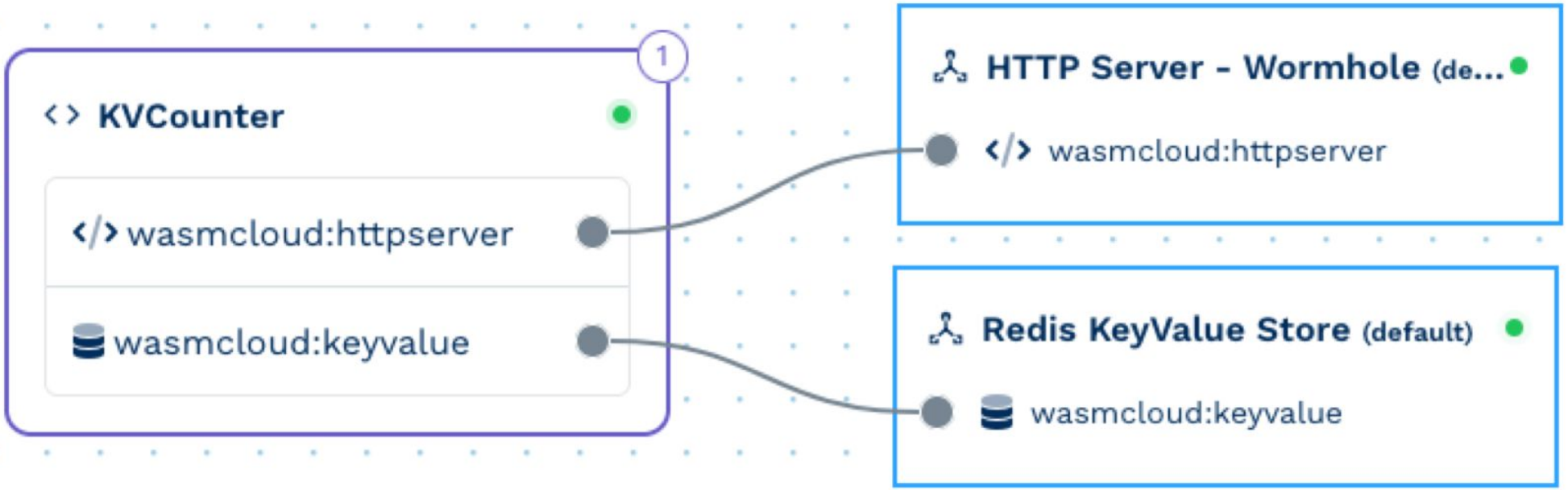
What tech do we use?

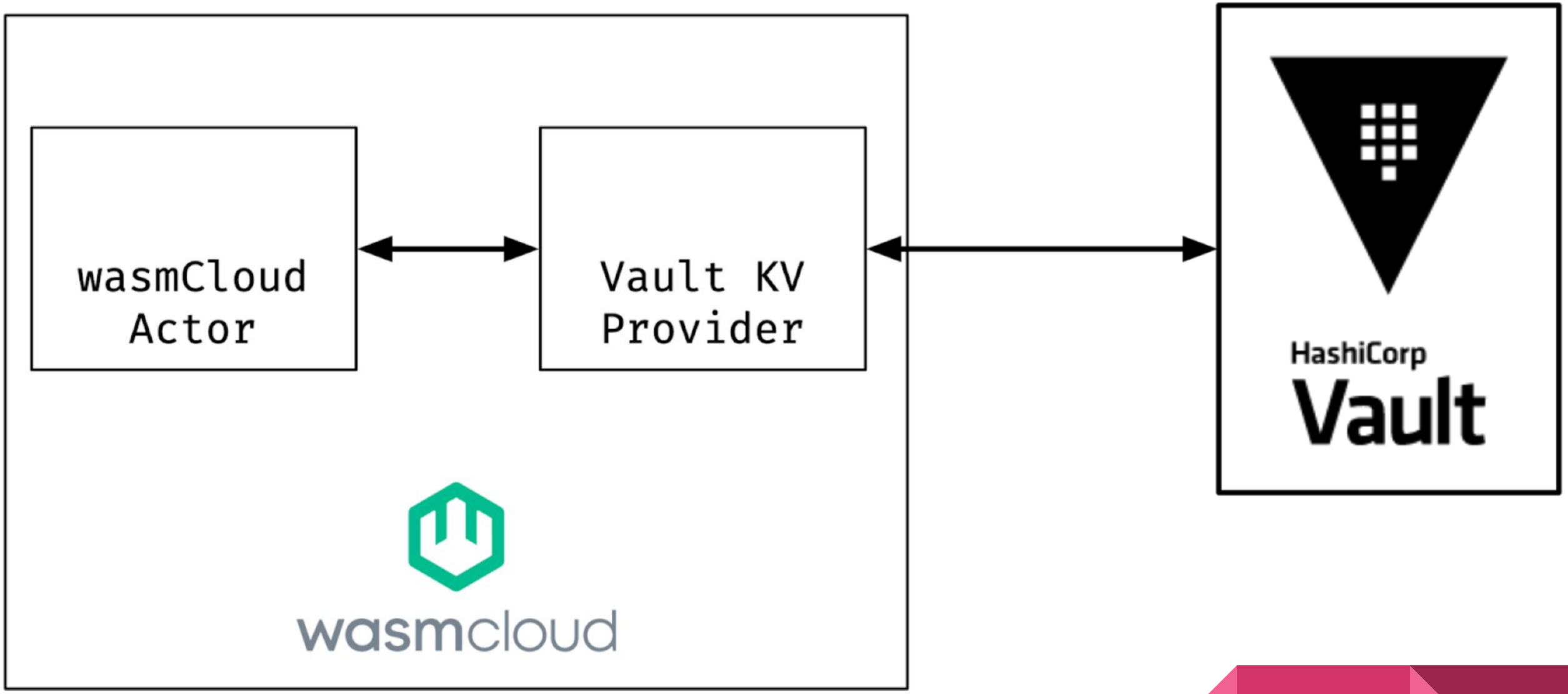
- Wasm, obvs
- OpenTelemetry
- Communication: NATS
- Data stores: Vault, Redis
- AWS (and many other cloud things)
- Infra: Nomad (with our own custom task driver!), Consul

Deep Dive

Data Store Abstractions

- Removing non-functional requirements was a huge benefit
- We started storing most data in Redis, but that isn’t the best place to store things like secrets

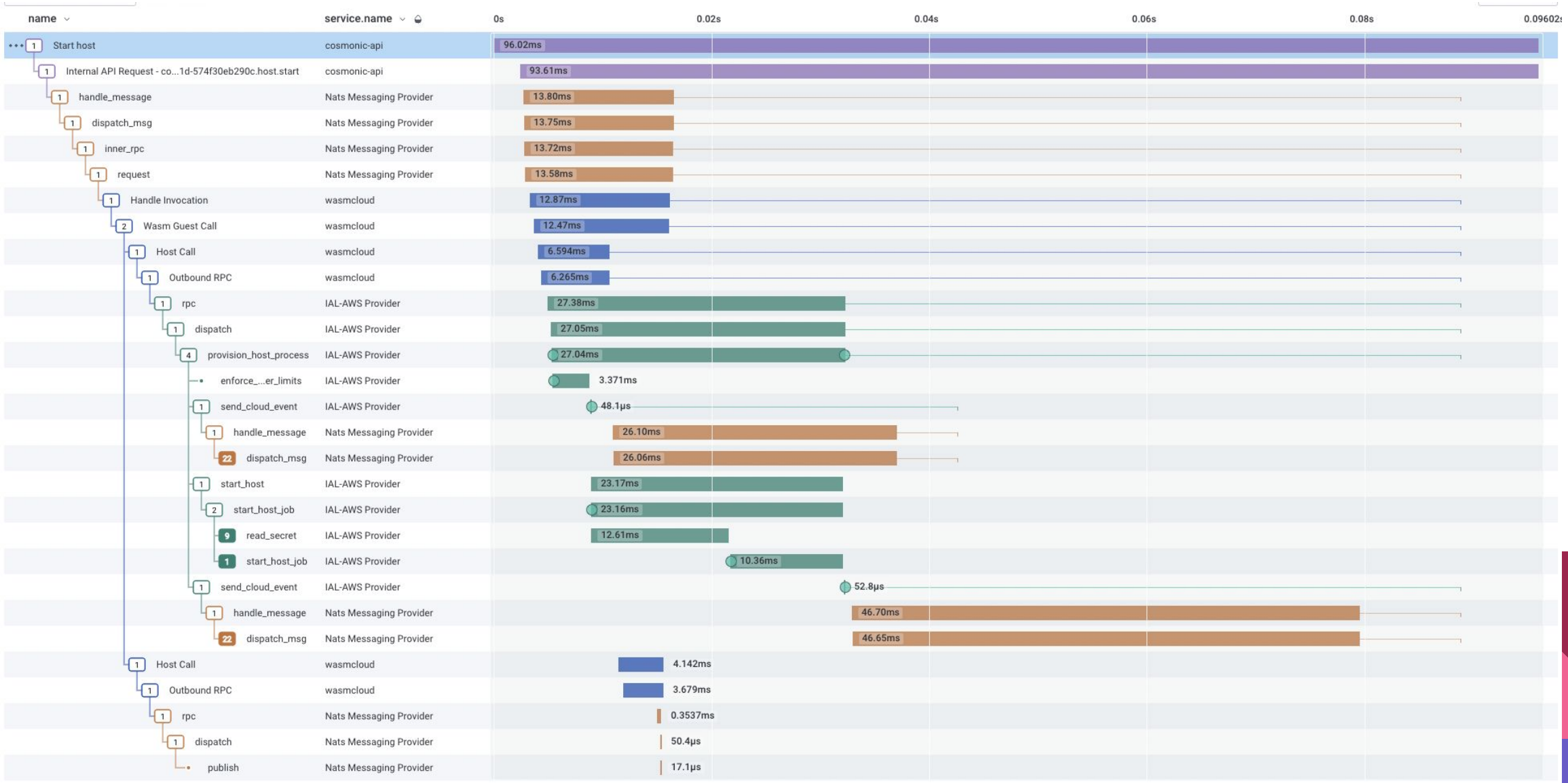

1
KVCounter
wasmcloud:httpserver
wasmcloud:keyvalue
HTTP Server - Wormhole (de...
wasmcloud:httpserver
Redis KeyValue Store (default)
wasmcloud:keyvalue


wasmCloud Actor
Vault KV Provider
HashiCorp
Vault
wasmcloud

Debugging and getting saved by Tracing

- wasmCloud components are designed to be distributed, even if they all run on the same machine
- Nondescript error messages hit us hard
- To add tracing to our platform, we needed to support it in open source first

An infrastructure provision trace



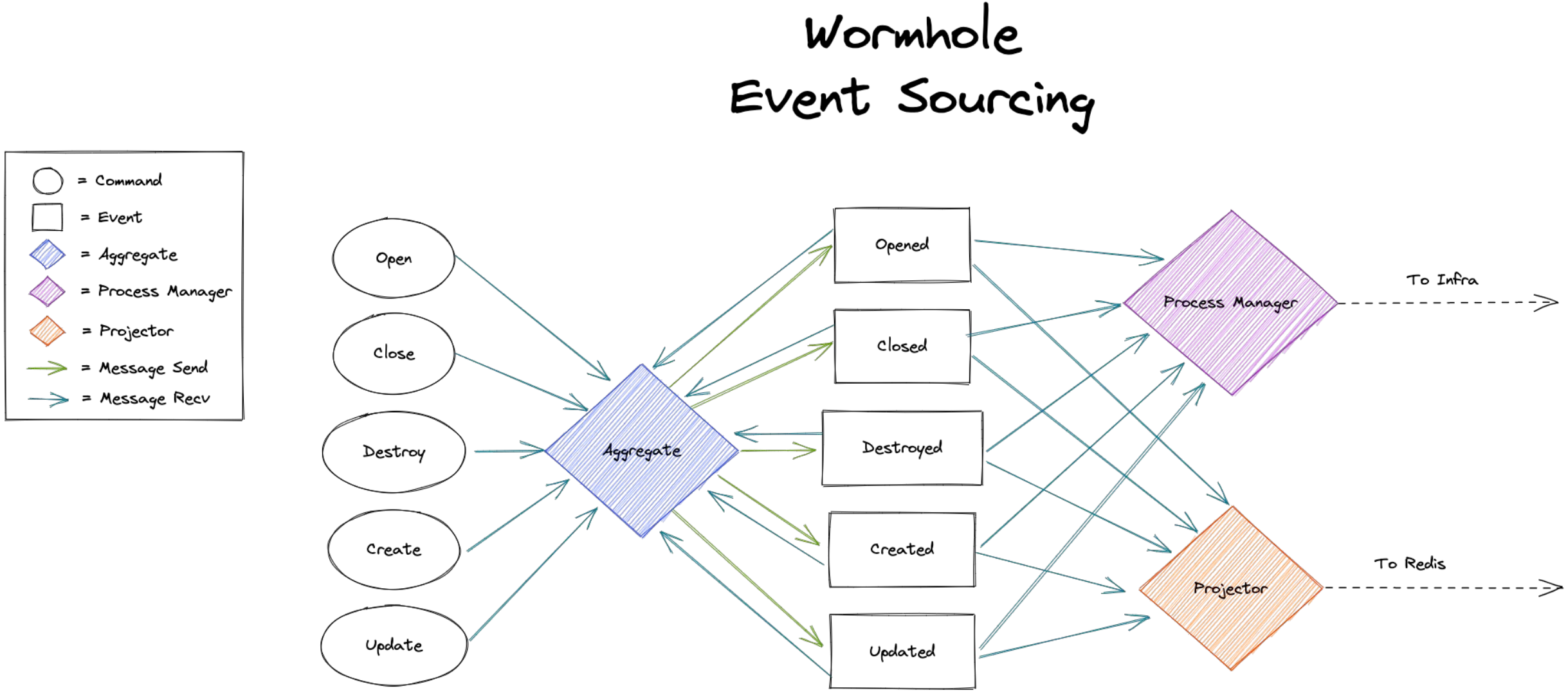
Speed and Scale

- Claim: WebAssembly is highly scalable and runs at a near-native speed
- Conclusion: ✅
- All performance bottlenecks and optimizations have been elsewhere in the stack

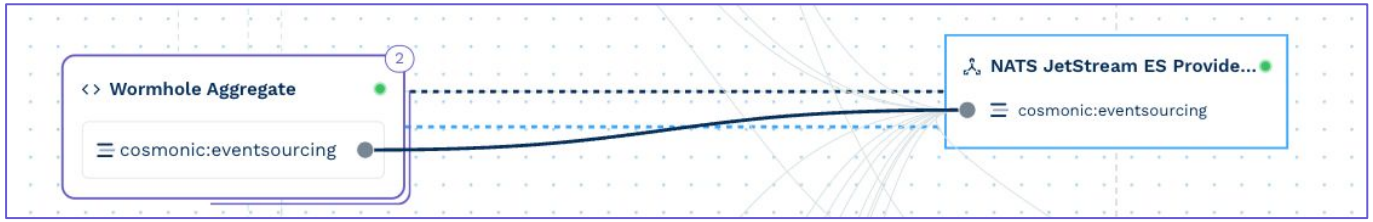
Event Sourcing and Reactive Programming

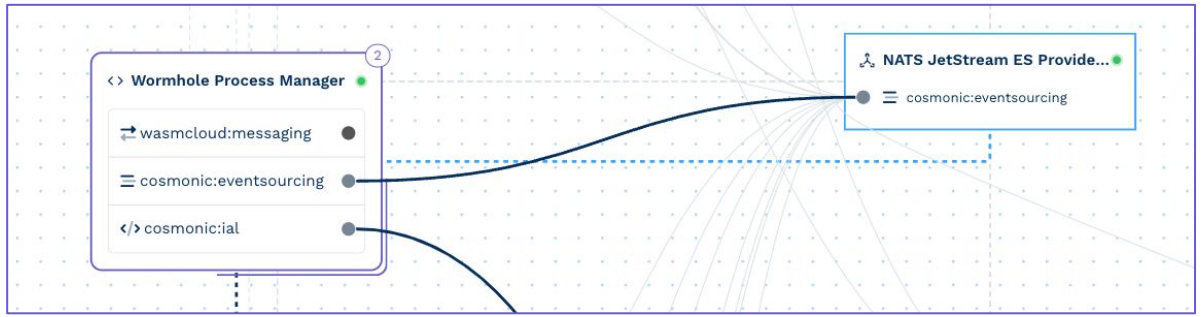
- Wasm is *really* good at this
- wasmCloud makes this even easier because you don’t have to decide on a specific source for your messages (e.g. Kafka, Rabbit MQ, NATS, etc)
- Event Sourcing is an architecture pattern that treats all state as a log and divides responsibility across different logical components called aggregates, process managers, and projectors

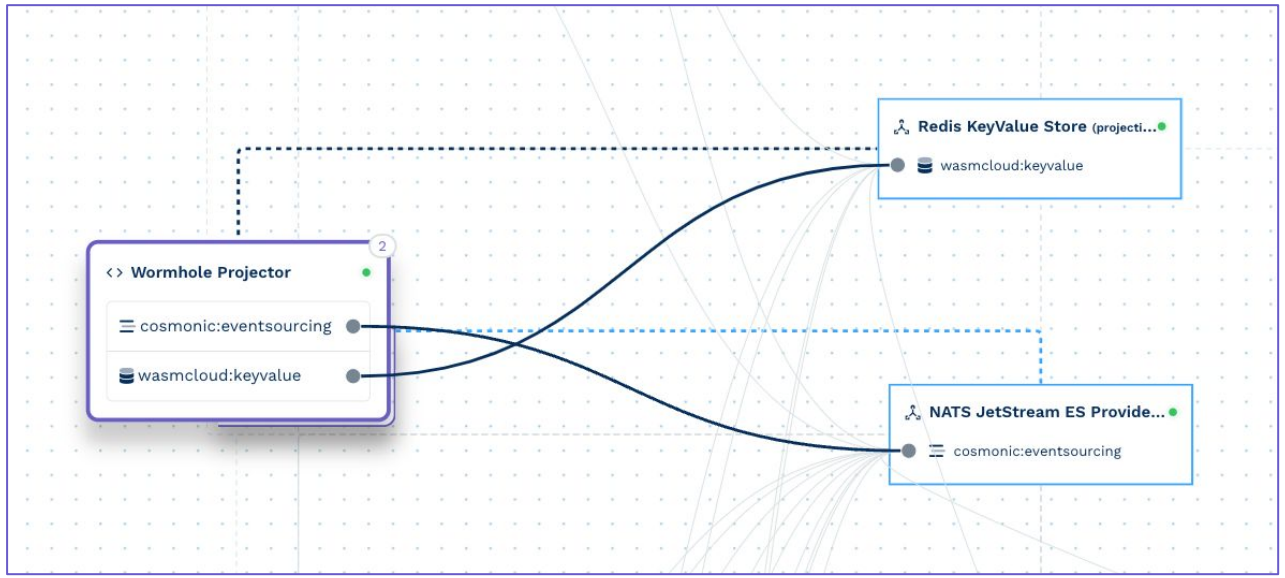
Event Sourcing and Reactive Programming



Event Sourcing and Reactive Programming







Abstracting over Infrastructures

- Provisioning wasmCloud on different infrastructure changes and isn’t platform agnostic like WebAssembly is
- Abstracting infrastructure is hard, so we kept it to a minimum (and still did too much)
- We needed the functionality of a scheduler

Defense in Depth

- We run untrusted code and need to have multiple tiers of defense
  - Wasm itself
  - wasmCloud host security
  - NATS security
  - Allow/deny rules
  - Firecracker

Lessons Learned

Welcome to the Bleeding Edge

- As with any new technology, you have to be ready to deal with changes
- However, you get a lot of really nice benefits from doing it
- We built a whole platform on top of it, so hopefully that should inspire some confidence

Wasm Pros vs Cons

Pros

- Speed and size is incredible
- Composability is king
- Deny-by-default security
- Flexible deployment options
- No Dockerfiles, Charts, Services Meshes, CRDs, etc

Cons

- Breaking changes are still likely
- Some things still just don’t work with Wasm (like capability providers) and you have to work around them
- The standards move slowly

wasmCloud-Specific Lessons

- Contract based development saved us so much time
- wasmCloud Actors + reactive programming == 🔥
- We really wish we could make providers Wasm *right now*
- Having the multiple layers of defense gave us a lot more confidence with multitenancy

What happens next?

For Cosmonic

- We will keep being our own Customer 0
- Each new feature means more Wasm, and we'll be sharing our learnings with the community
- We plan on driving forward work in the Wasm standards, especially things that push forward the ability to compile providers to Wasm

For Wasm

- Component Model
- SIG Registry and Bindle
- Making networking even easier to use

References

- The Cosmonic Platform
  - https://cosmonic.com/
  - https://cosmonic.com/docs/
- https://github.com/wasmCloud/wasmCloud
- Additional resources
  - https://github.com/cosmonic/kubernetes-applier
  - https://github.com/wasmCloud/capability-providers
  - https://github.com/wasmCloud/interfaces

Thank you!