



WebAssembly and scale-to-zero for more sustainable computing

Stuart Harris, Red Badger

Green Code London

20th March 2025



Hi, I'm Stu

- Software engineer
- Founder and Chief Scientist at Red Badger
- @stuartharris





About us

Red Badger is the Digital Product transformation consultancy

We help modern enterprises continuously evolve their products and services. We craft digital products customers love, build next generation platforms and embed new digital capabilities



15

Years old, founded 2010

c.100

People

c.90%⁺

Permanent, London team



TESCO

FINANCIAL TIMES

sky

BBC

D
DOW JONES

asos

Levi's®

HSBC ◀▶

LLOYDS BANK

JLT

**Fidelity™
INTERNATIONAL**

**LME
An HKEX Company**

TANDEM

chopin

FORTNUM & MASON
EST 1707

**CAMDEN
MARKET**

**BRITISH
COUNCIL**

CHASE ◀▶

**PRIDE IN
LONDON**

~~ANTHONY
NOLAN~~

ATKINS

cartrawler



MHRA

News UK

EQUIFAX

Nando's

Edelman

Santander

Ventures | **BARCLAYS**



What is **WebAssembly**?



Well, it's not (only) Web and it's not Assembly!

It is a bytecode (like Java bytecode or the Common Intermediate Language of .Net).

More formally — it's a binary instruction format for a stack-based virtual machine.



How is it different?

Simple

Possibly the simplest virtual machine we have. Only has **4 types** (i32/64 and f32/64). No baked-in OOP concepts (like JVM). No coupling to APIs, the DOM, or screen-space (like Java applets).

Secure

Designed to run untrusted code in the browser. **Deny-by-default** sandbox ensures code cannot, itself, run *any* side effects.

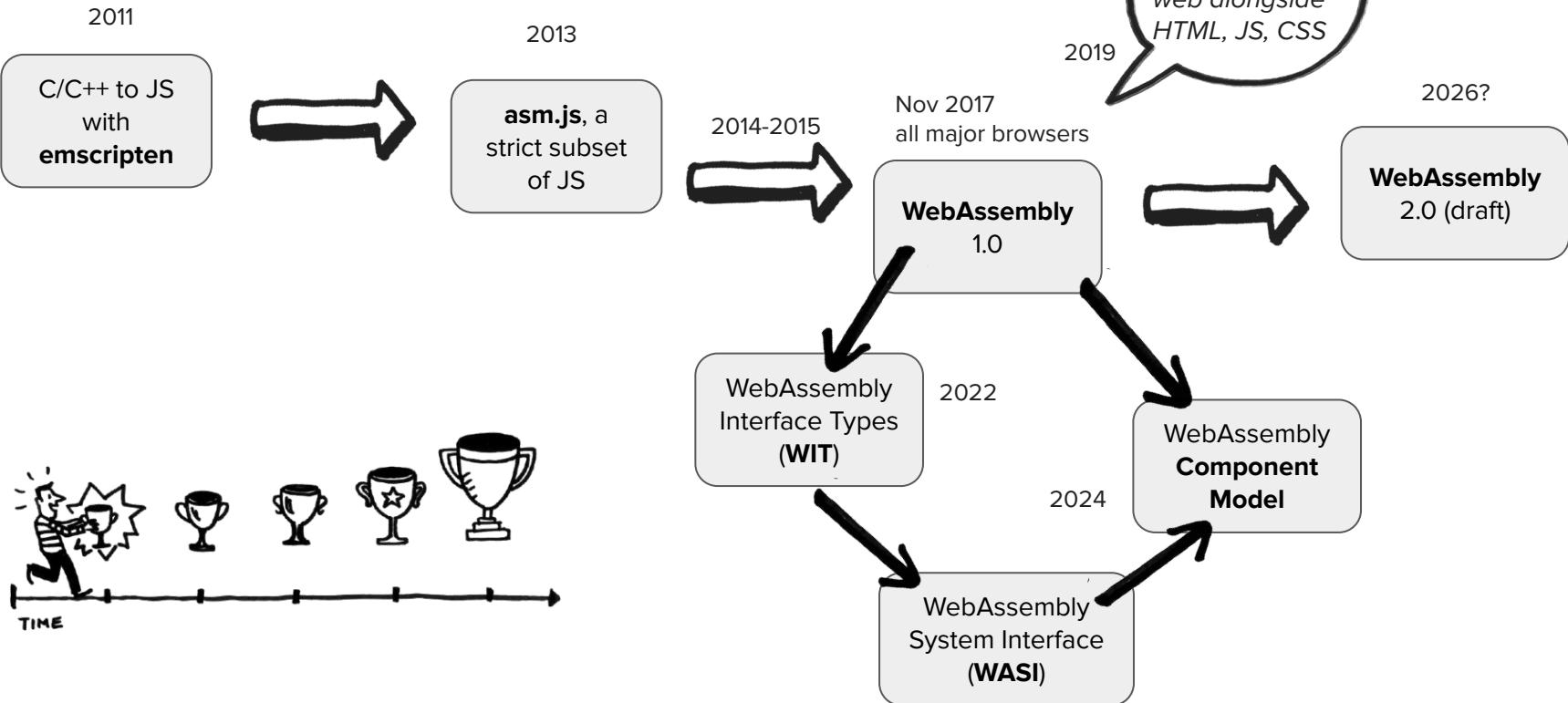
Speedy

Designed to run code at **native** speeds. Lightweight. No runtime or garbage collection (although there is a spec for that). Streaming compilation.





A brief History





Demo

What is the simplest WebAssembly module
we could create?





WebAssembly Text

The screenshot shows a code editor window titled "wasm" with the file "add.wat" open. The code is written in WebAssembly Text (WAT) format:

```
1 (module
2   (func (export "add") (param $x i32) (param $y i32) (result i32)
3     local.get $x
4     local.get $y
5     i32.add))
```

The editor has a toolbar at the top with icons for back, forward, and search, and a status bar at the bottom indicating "6:1 WebAssembly Text Format".

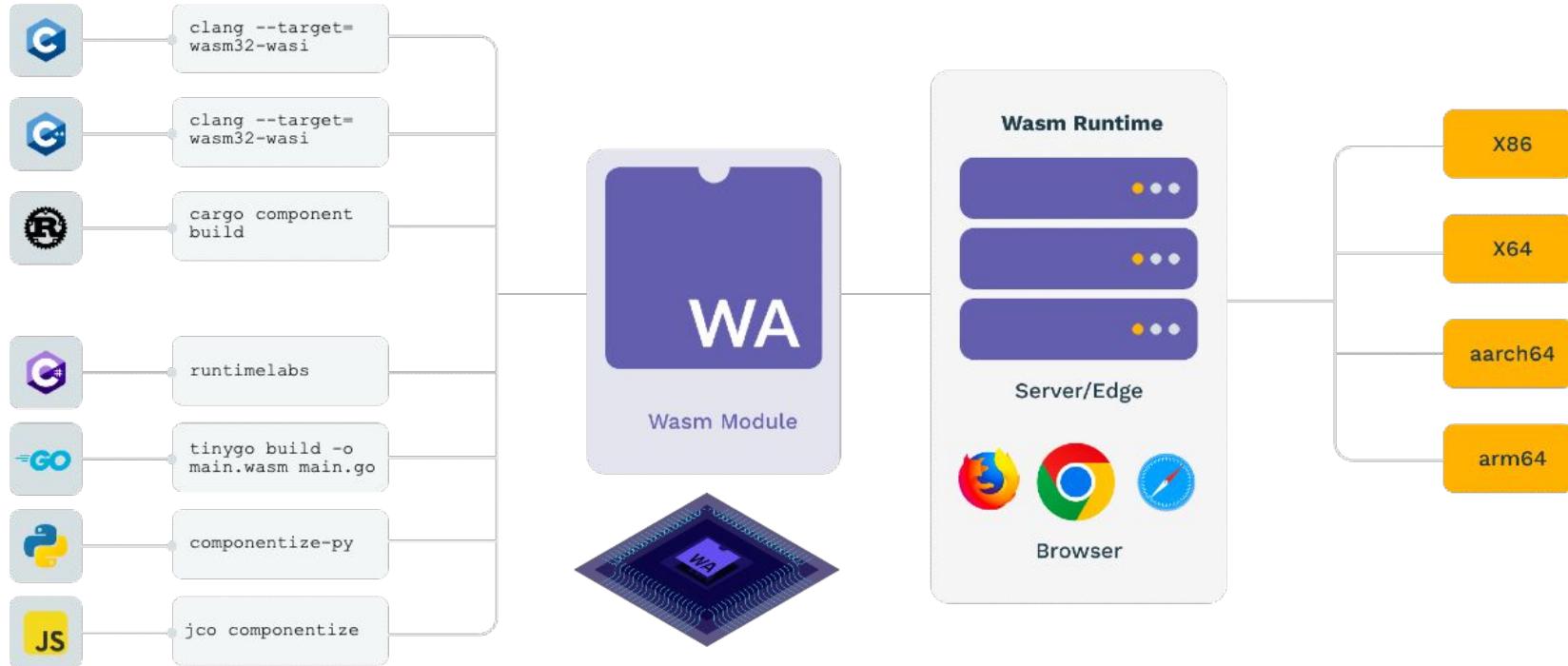
The screenshot shows a terminal window with the following session:

```
stuartharris ~/tmp/wasm 11:36
→ wasmer add.wat -i add 22 44
66
```

The terminal is located in a folder named "/t/wasm".



WebAssembly Modules

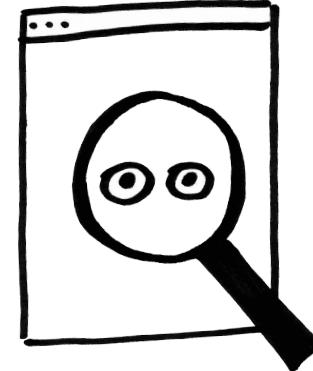




Demo

Let's use a *real programming language* to create a WebAssembly module on MacOS.

... and then let's run that *same binary* on a Linux machine!





Creating a WebAssembly Module in Rust

```
stuartharris ➜ /tmp/wasm ⑧ 11:39
→ cargo new hello-world
  Creating binary (application) `hello-world` package
note: see more `Cargo.toml` keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

stuartharris ➜ /tmp/wasm ⑧ 11:39
→ cd hello-world/

stuartharris ➜ .../hello-world ⑧ master ? ⑧ v1.85.1 ⑧ 11:40
→ bat ./src/main.rs

File: ./src/main.rs
1 fn main() {
2     println!("Hello, world!");
3 }

stuartharris ➜ .../hello-world ⑧ master ? ⑧ v1.85.1 ⑧ 11:40
→ cargo build --release --target wasm32-wasip2
  Compiling hello-world v0.0.1 (/private/tmp/wasm/hello-world)
  Finished `release` profile [optimized] target(s) in 1.25s

stuartharris ➜ .../hello-world ⑧ master ? ⑧ v1.85.1 ⑧ 11:40
→ eza -la target/wasm32-wasip2/release/hello-world.wasm
.rw-r--r--@ 87k stuartharris 20 Mar 11:40 target/wasm32-wasip2/release/hello-world.wasm

stuartharris ➜ .../hello-world ⑧ master ? ⑧ v1.85.1 ⑧ 11:40
→ wasmtime target/wasm32-wasip2/release/hello-world.wasm
Hello, world!
```



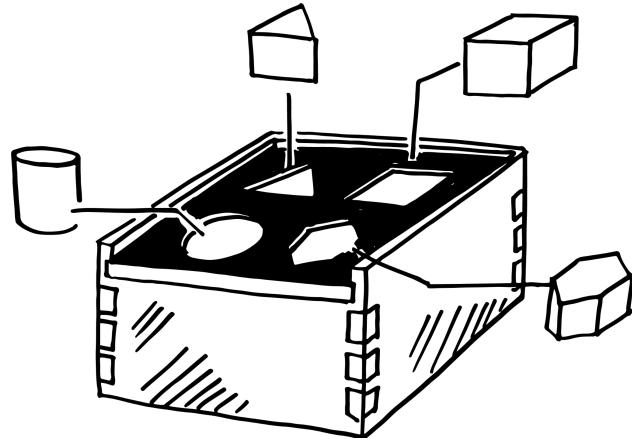
WebAssembly Components

WebAssembly Interface Types

WASI preview 1 (modules)

WASI preview 2 (components)

- Polyglot
- Canonical ABI, with static and dynamic linking
- Capability based security
- Bindings generation — e.g. wasm-bindgen
- Interface virtualisation — a component can't tell if the other side is another component or the host
- Shared nothing architecture, with resources



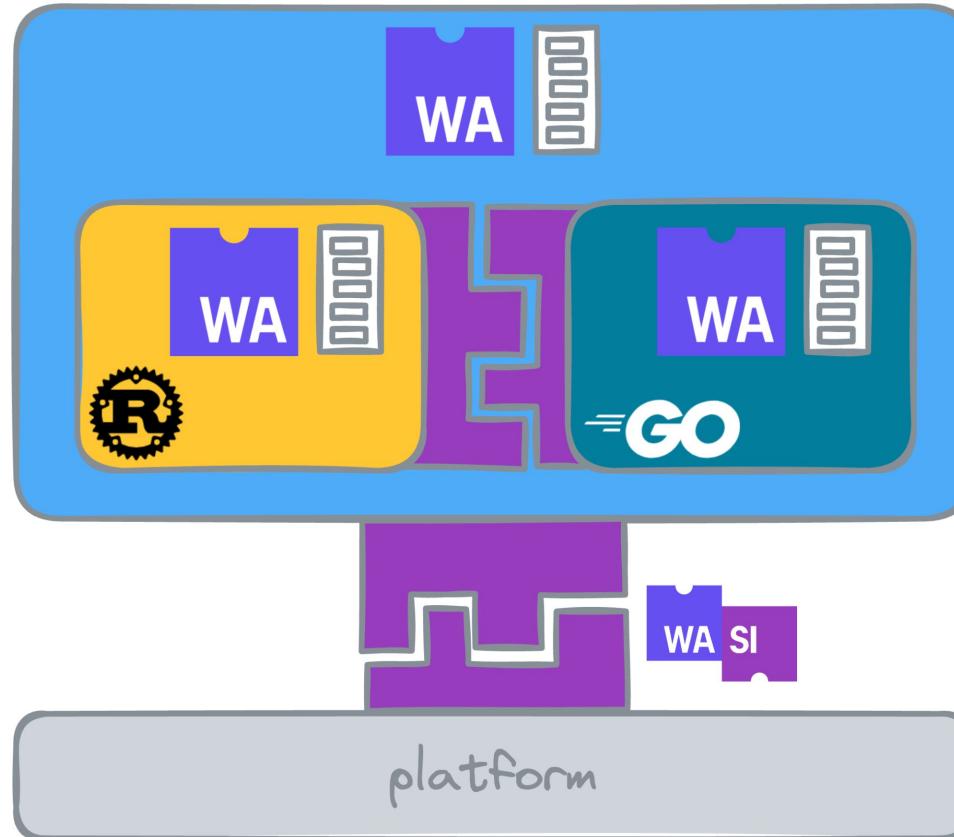


WebAssembly Components



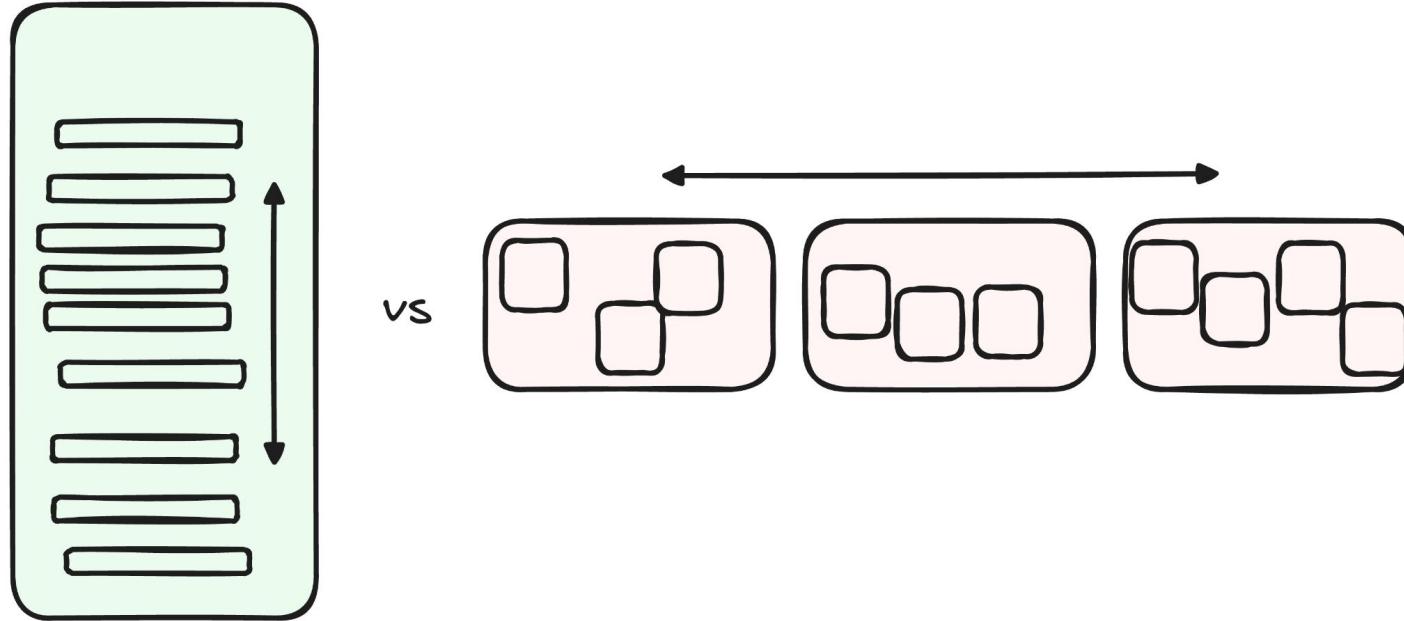


WebAssembly Components



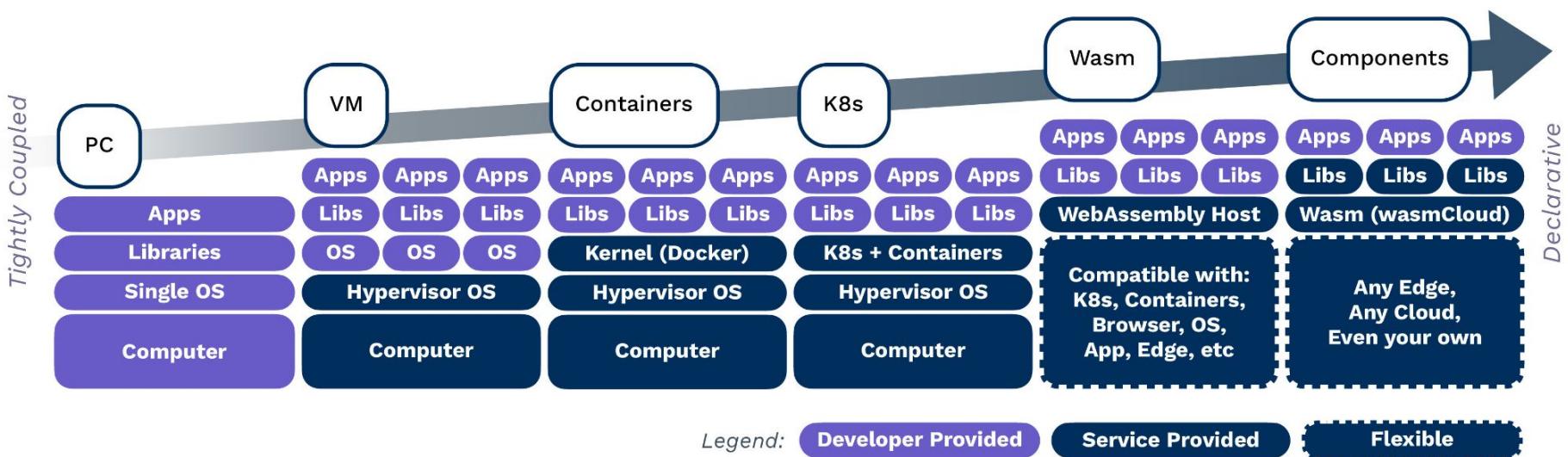


Scaling Vertically and Scale-to-zero



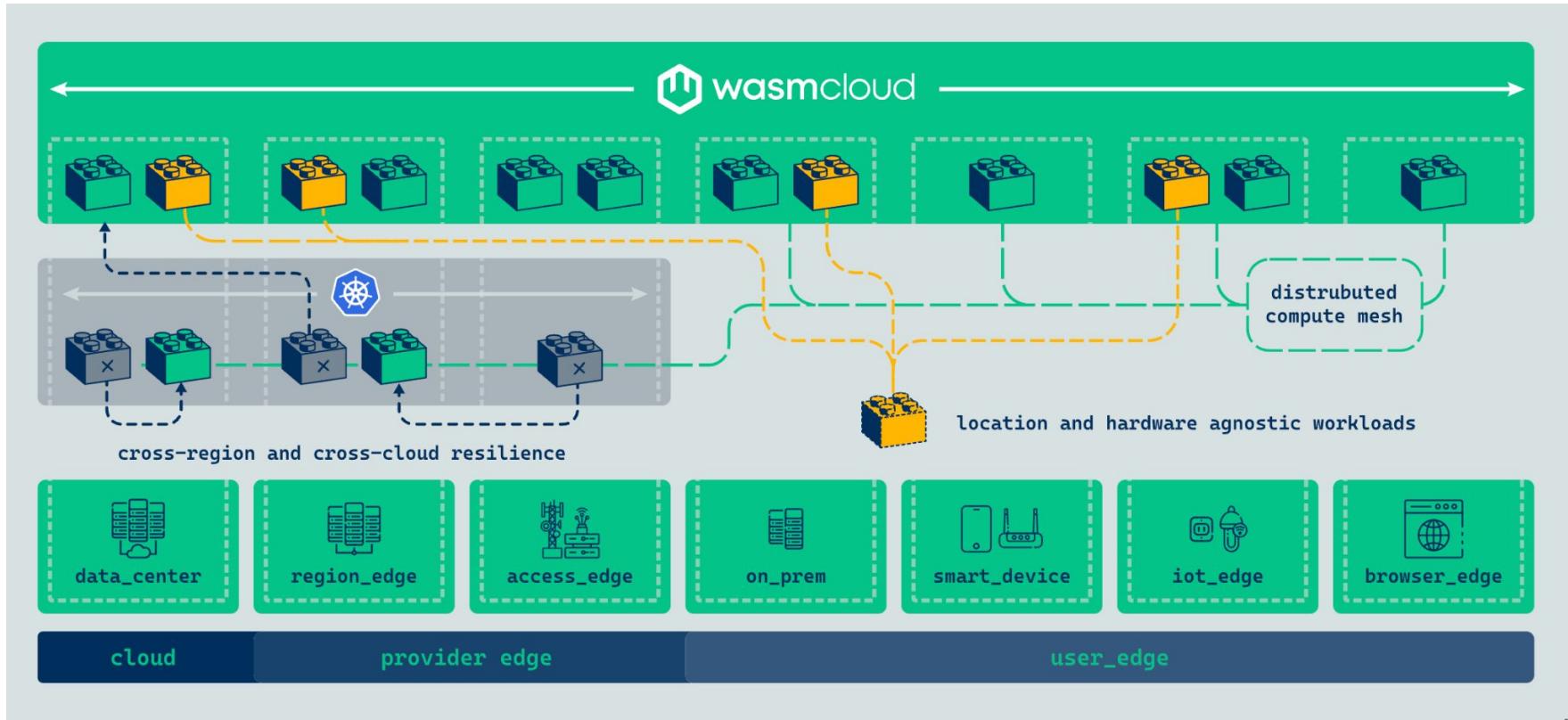


Platform evolution





The **wasmCloud** platform





Build

Faster Development Cycles

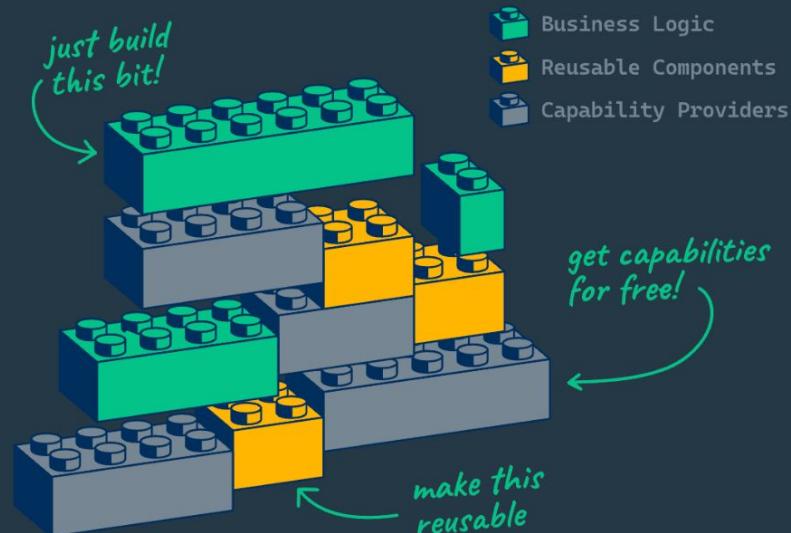
Leverage reusable, polyglot, Wasm components on a reliable, distributed platform.

Centrally Maintainable Apps

Reusable, version-controlled components empower platform teams to maintain thousands of diverse apps centrally.

Integrate with Existing Stacks

wasmCloud has first-tier support for Kubernetes, AWS, Azure, GCP, Jenkins, Github Actions, ArgoCD, Backstage, Chainguard, Databases, Messaging, and more.





Compose

Development Without Lock-In

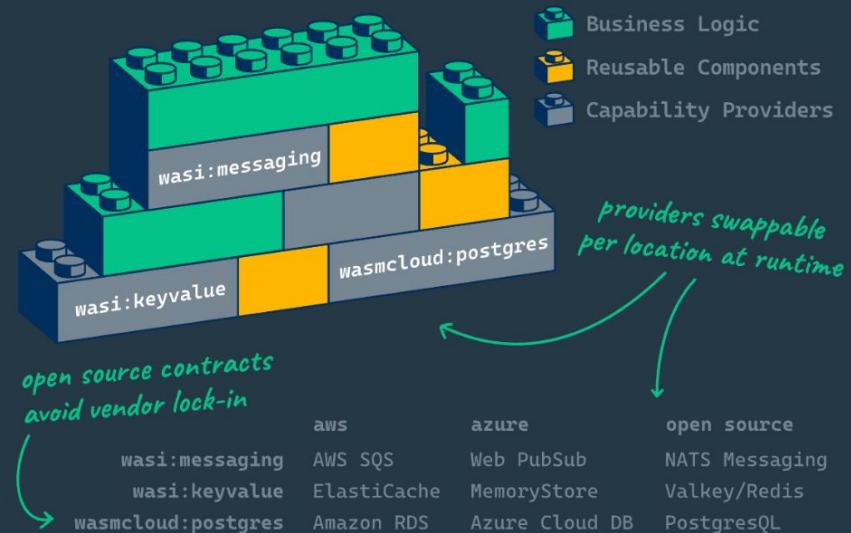
Define application dependencies at runtime via contract driven interfaces leveraging different vendors across deployments, dev, QA, or prod.

Truly Portable Apps

Run the same Wasm application across operating systems and architectures—no new builds required. Linux, MacOS X, Windows, ARM, x86, and more.

Custom Capabilities

Easily extend the secure wasmCloud host at runtime to support custom dependencies, hardware, or business contracts.





Run

Scale-to-Zero with Zero Cold Starts

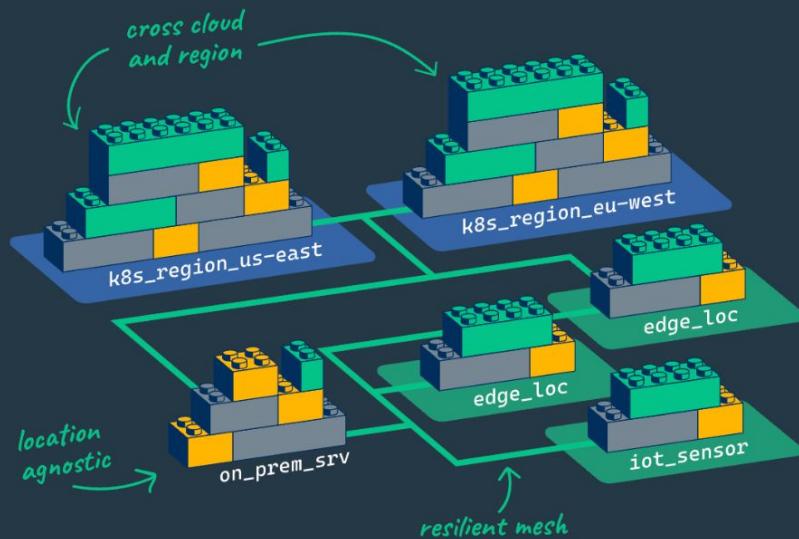
Sub-millisecond start times and vertical autoscaling means workloads scale to the demand.

Reliable, Fault-Tolerant Apps

Horizontal scaling with automated fail-over gives apps capability-level resiliency, reliability, and scalability.

Deploy Across Clouds

Close to your users, with local-first routing and at-most-once delivery, wasmCloud delivers cross-region, cross-cloud, and cross-edge capability-level resiliency to every deployment





WA

WebAssembly

“WebAssembly (abbreviated Wasm) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications.”

Performance

Simple stack-based virtual machine for executing code written in *any* language at near native speeds, with almost no overhead.

Safety

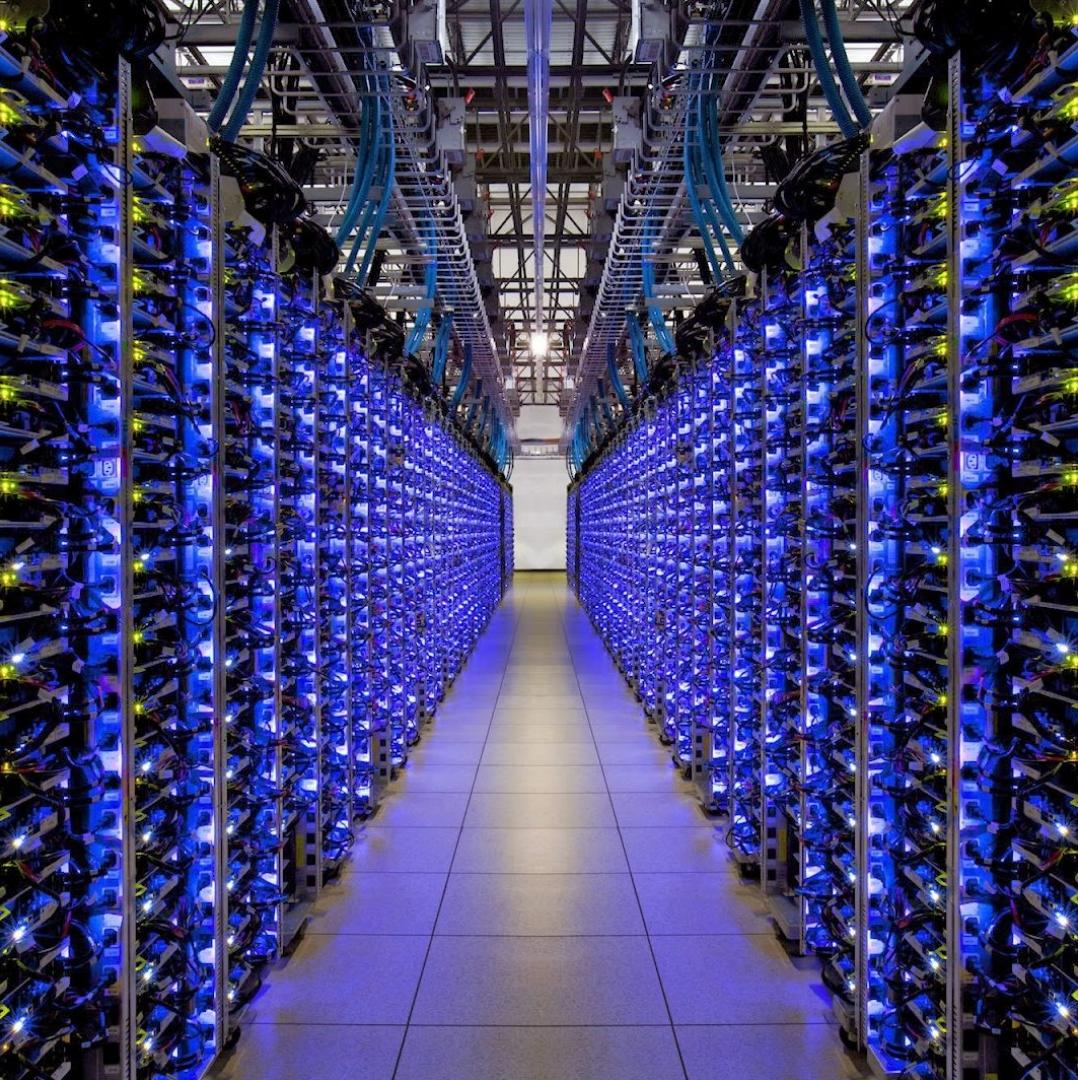
Designed for running untrusted code in the browser, Wasm’s sandbox is *essential* for running enterprise applications that are composed from open source software.

Portability

Portable across all machine architectures and operating systems, Wasm binaries are small and can start up instantly. Components are a *standard shape* and portable across platforms and clouds.

Data Centres in Ireland

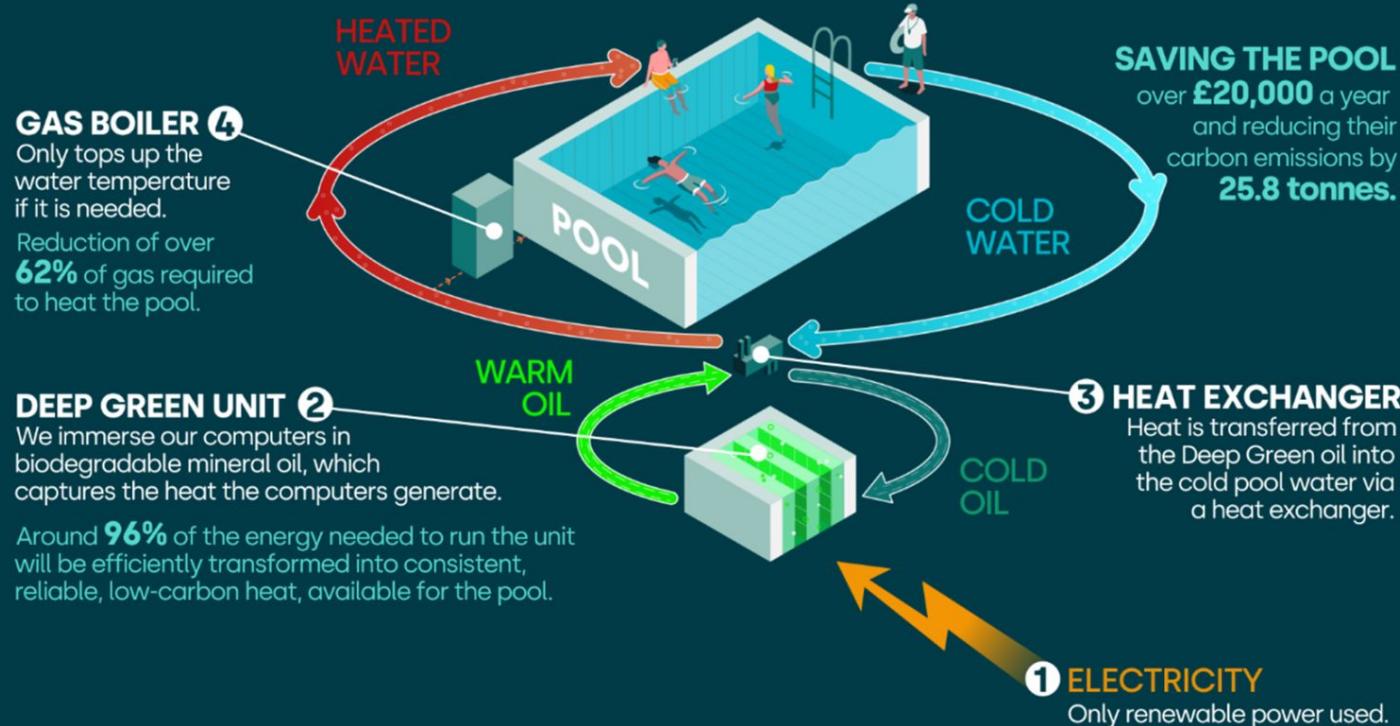
- **82** operational
- **14** under construction
- **52** planned [ref](#)
- Consume **21%** of the nation's electricity,
that's more than all urban homes (18%),
and projected to rise to **33%** within **3**
years [ref](#)
- **50.7%** fossil fuels in 2023, including
coal, peat and oil, and not including
imports [ref](#)





DEEP
GREEN

HOW DEEP GREEN HEATS A POOL





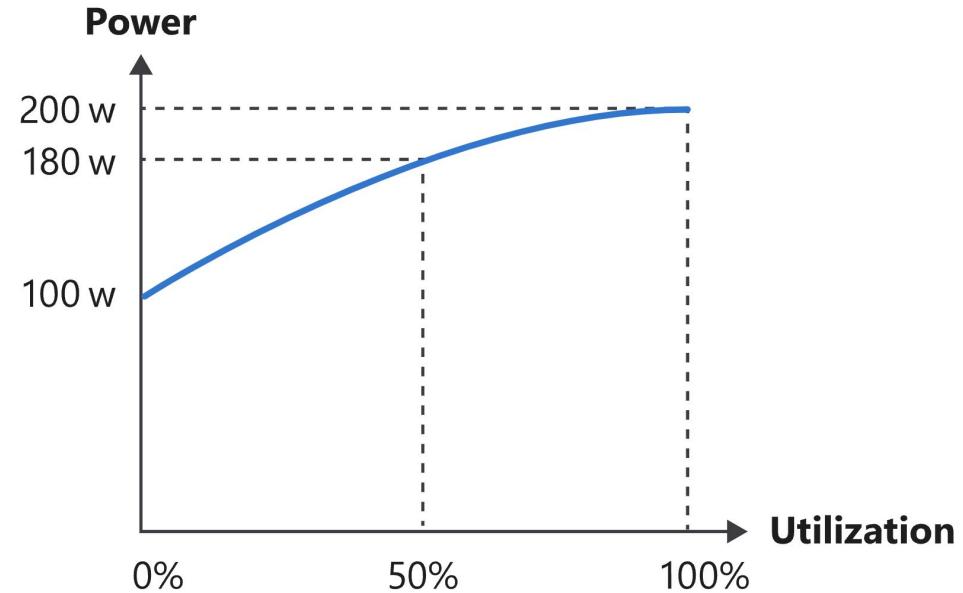
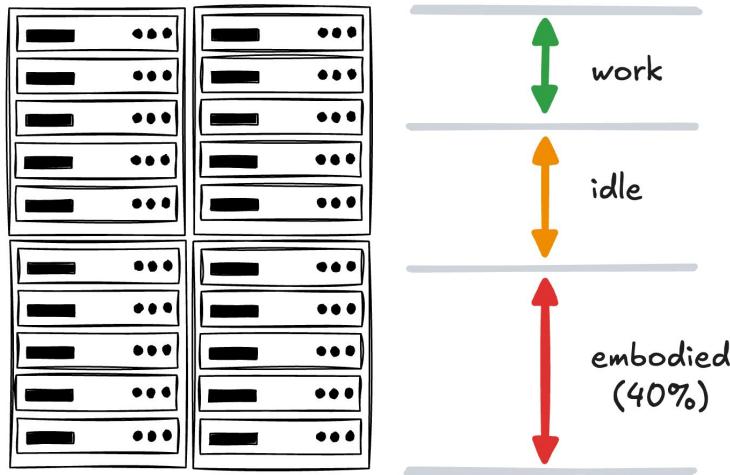
Equinix PA10





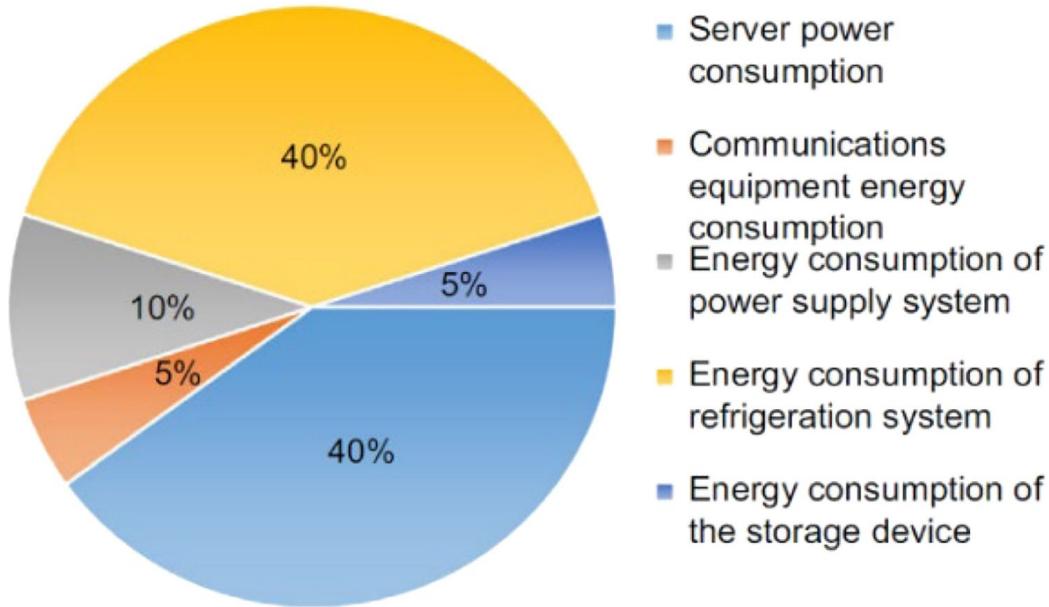
Carbon

Lifetime Carbon



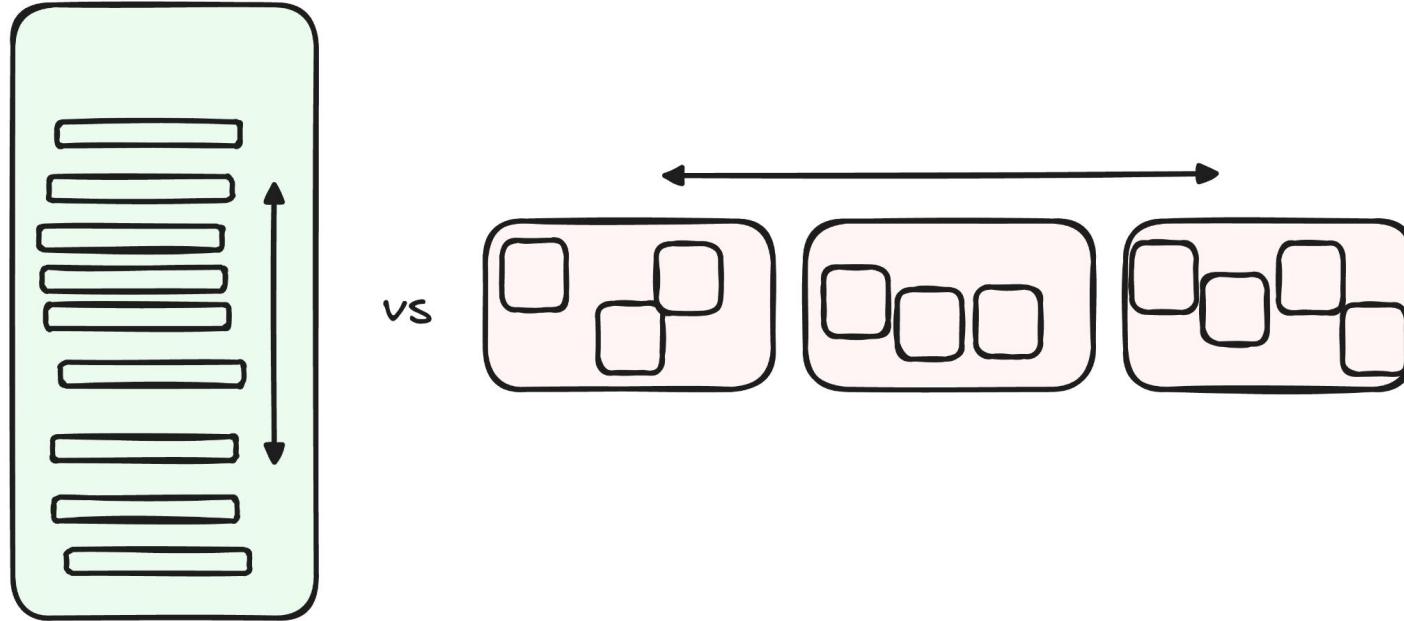


Cooling

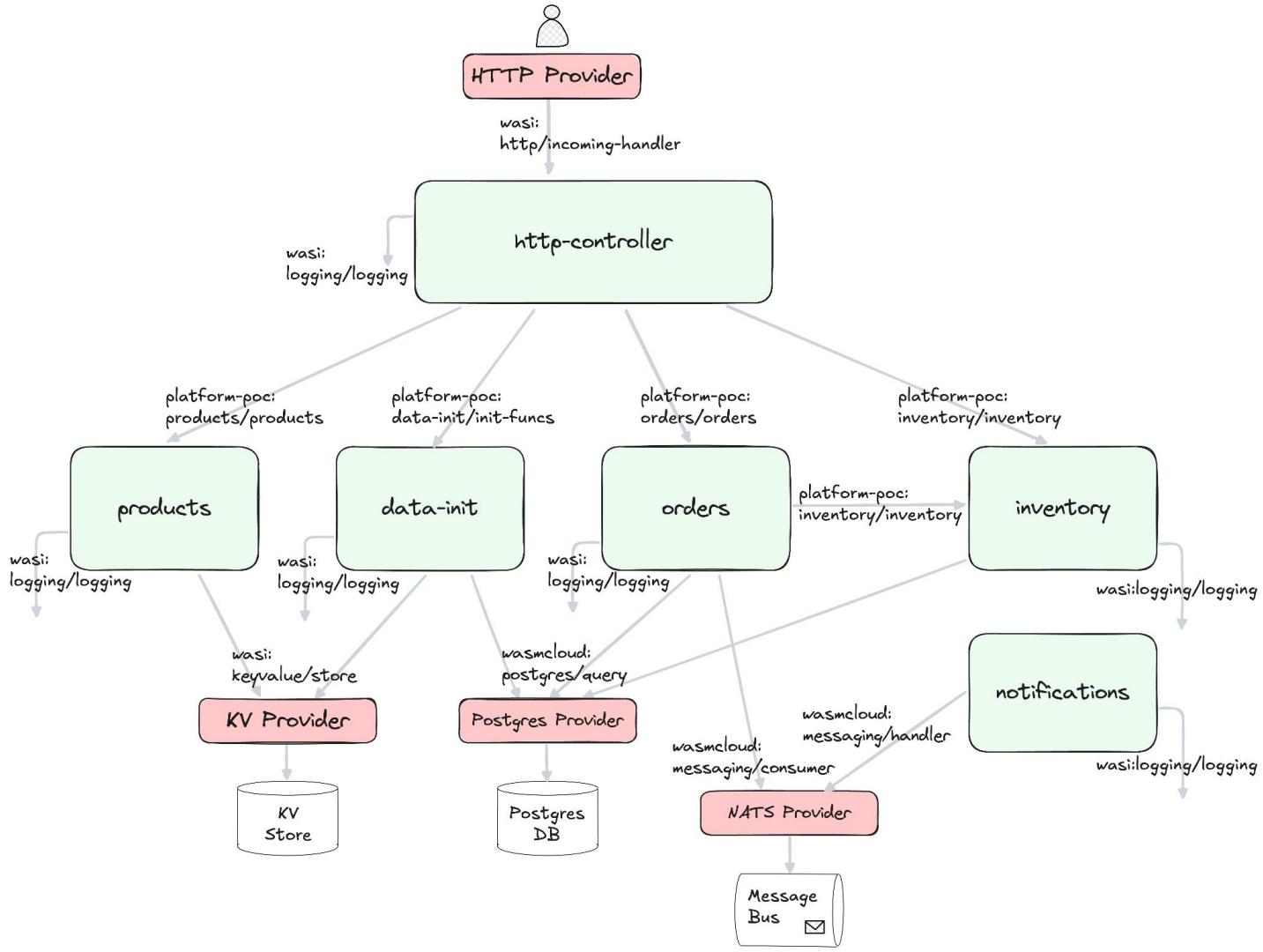




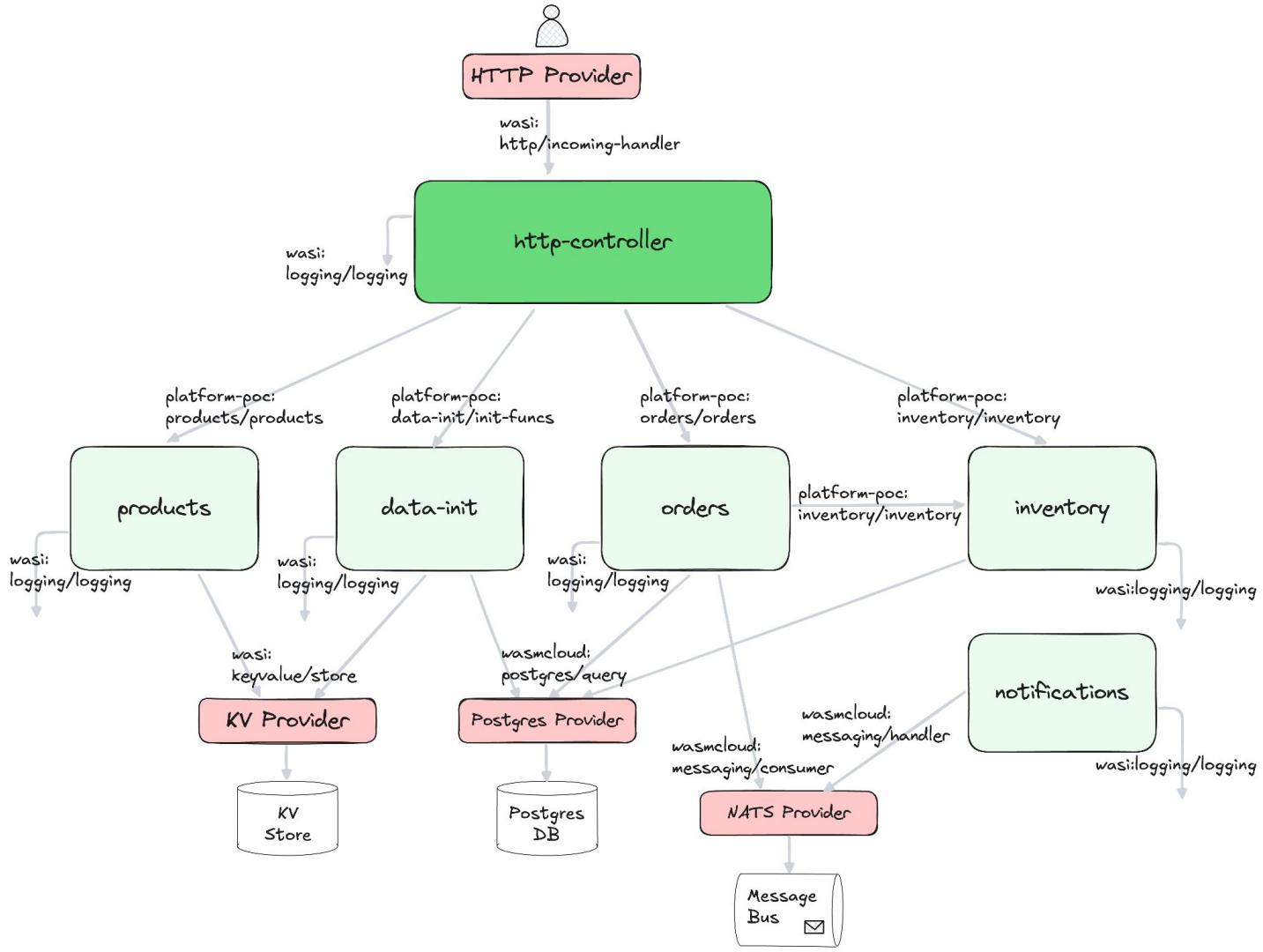
Scaling Vertically and Scale-to-zero



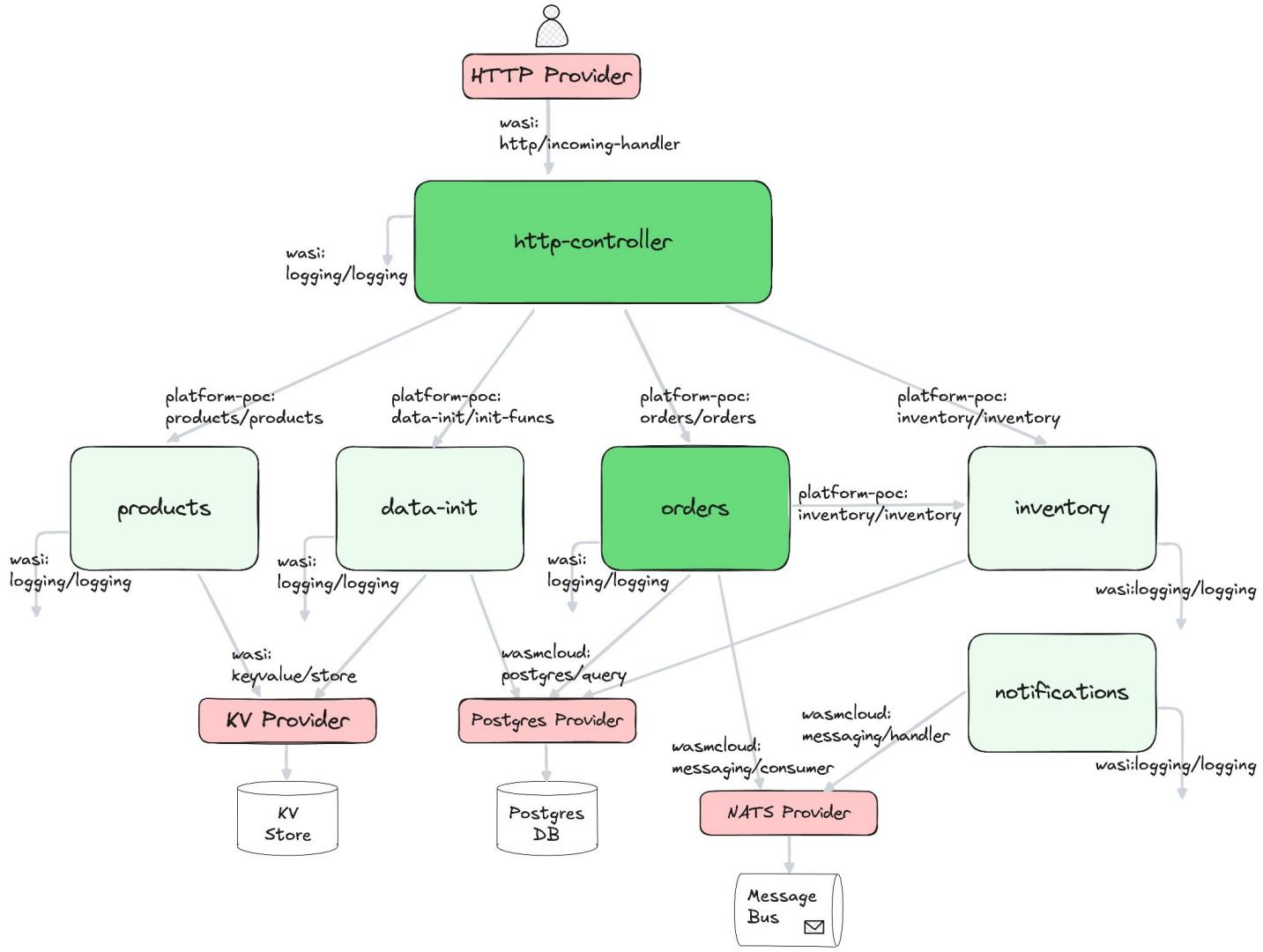
Example



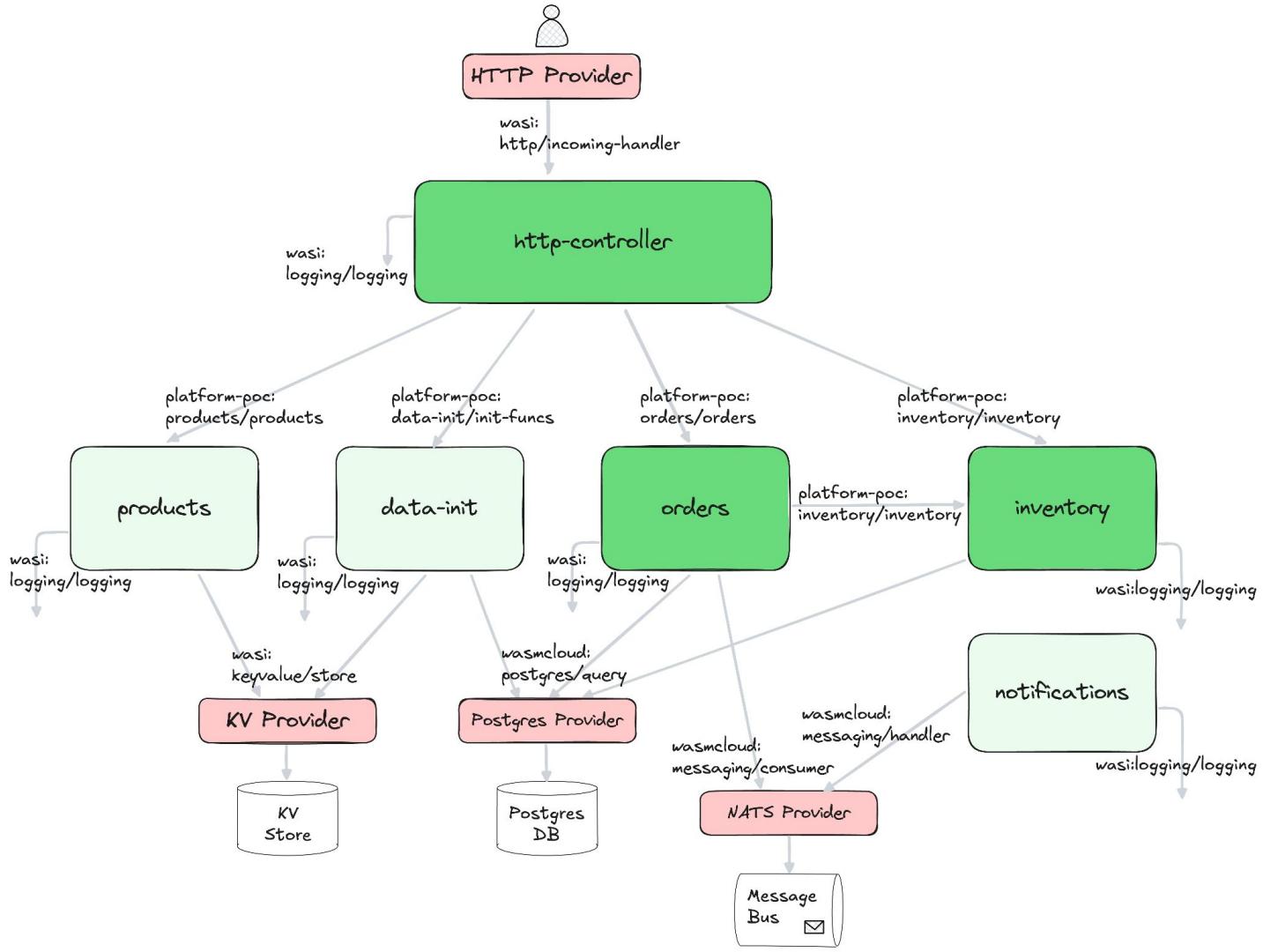
Example



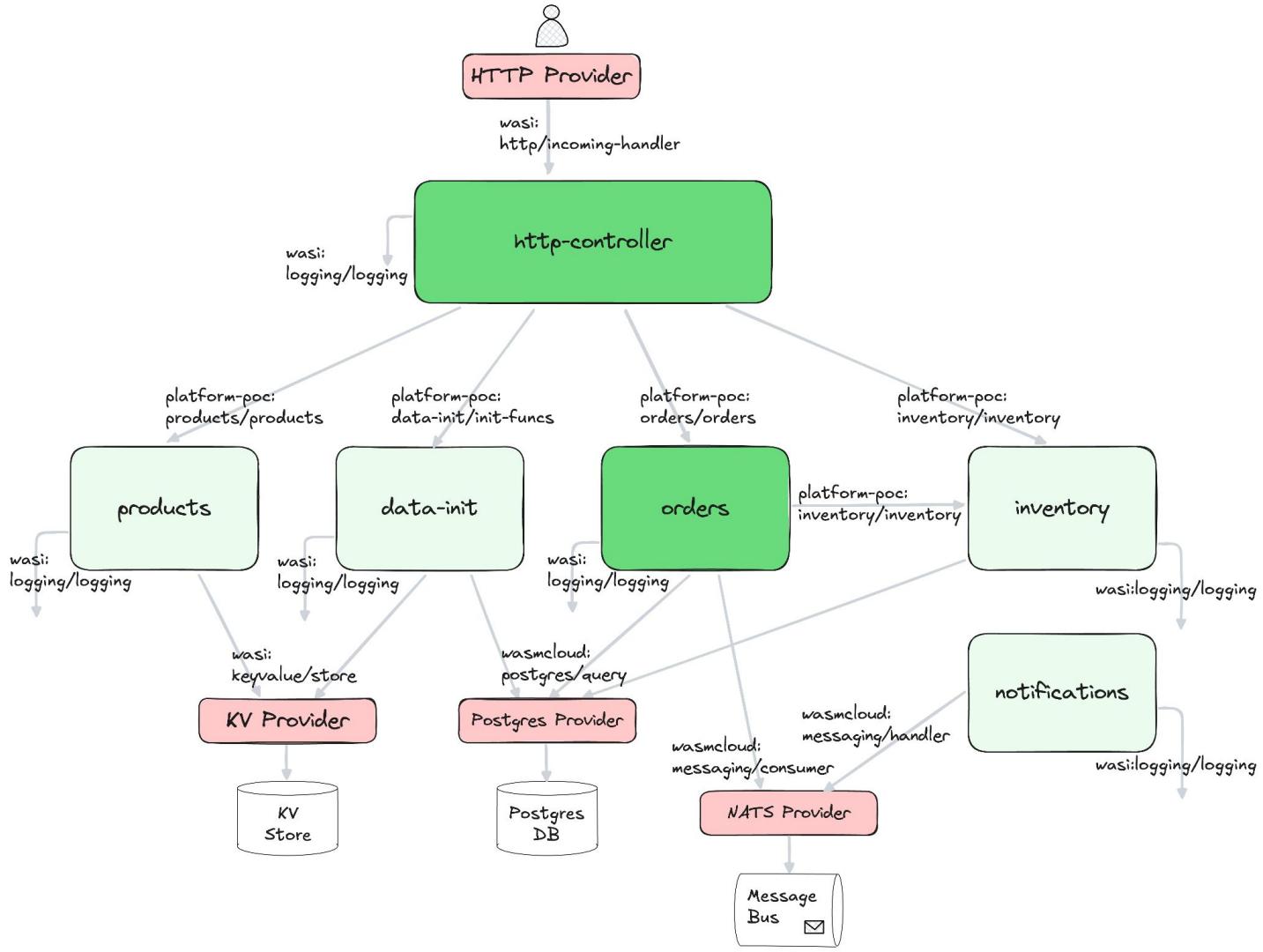
Example



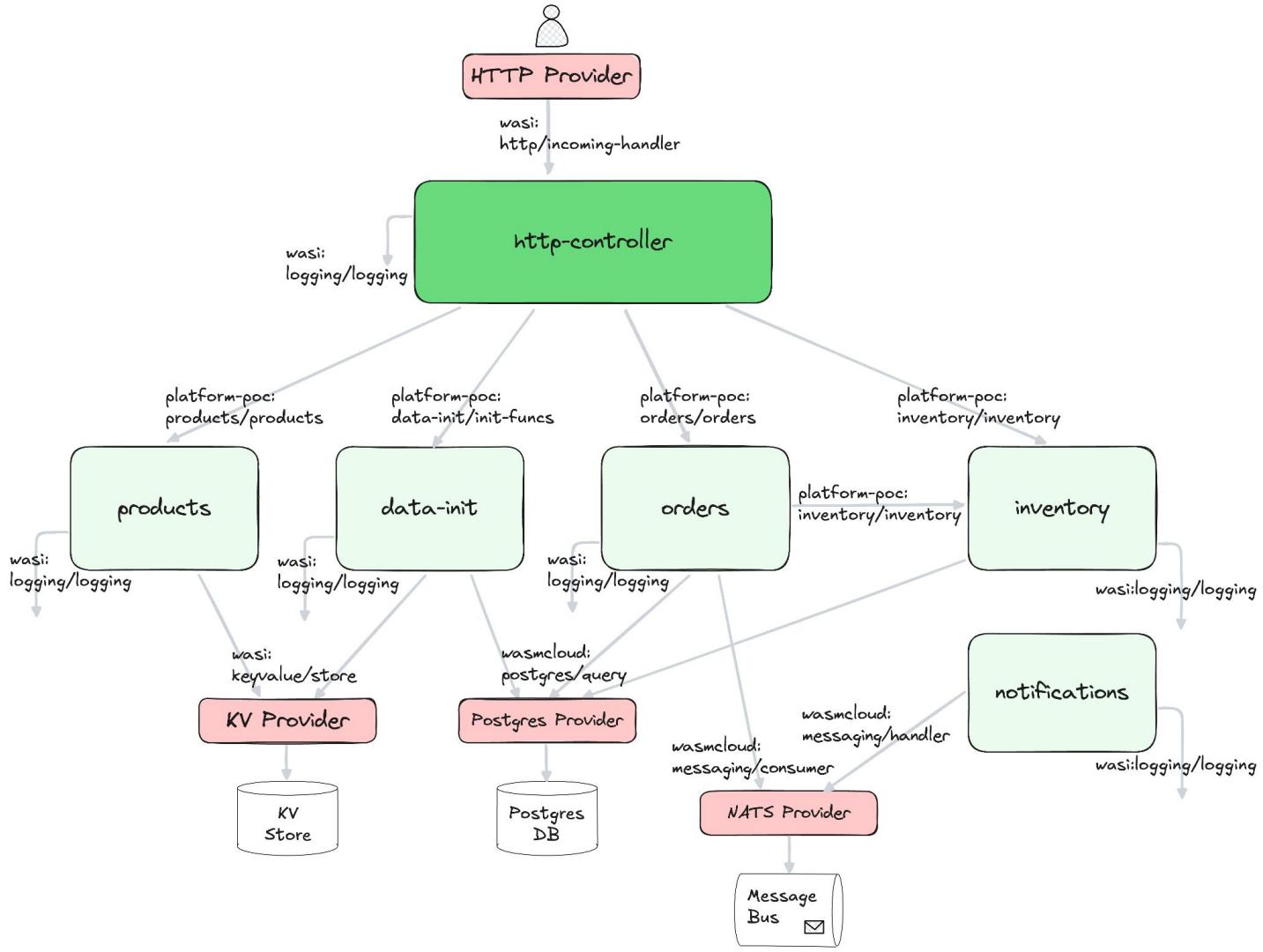
Example



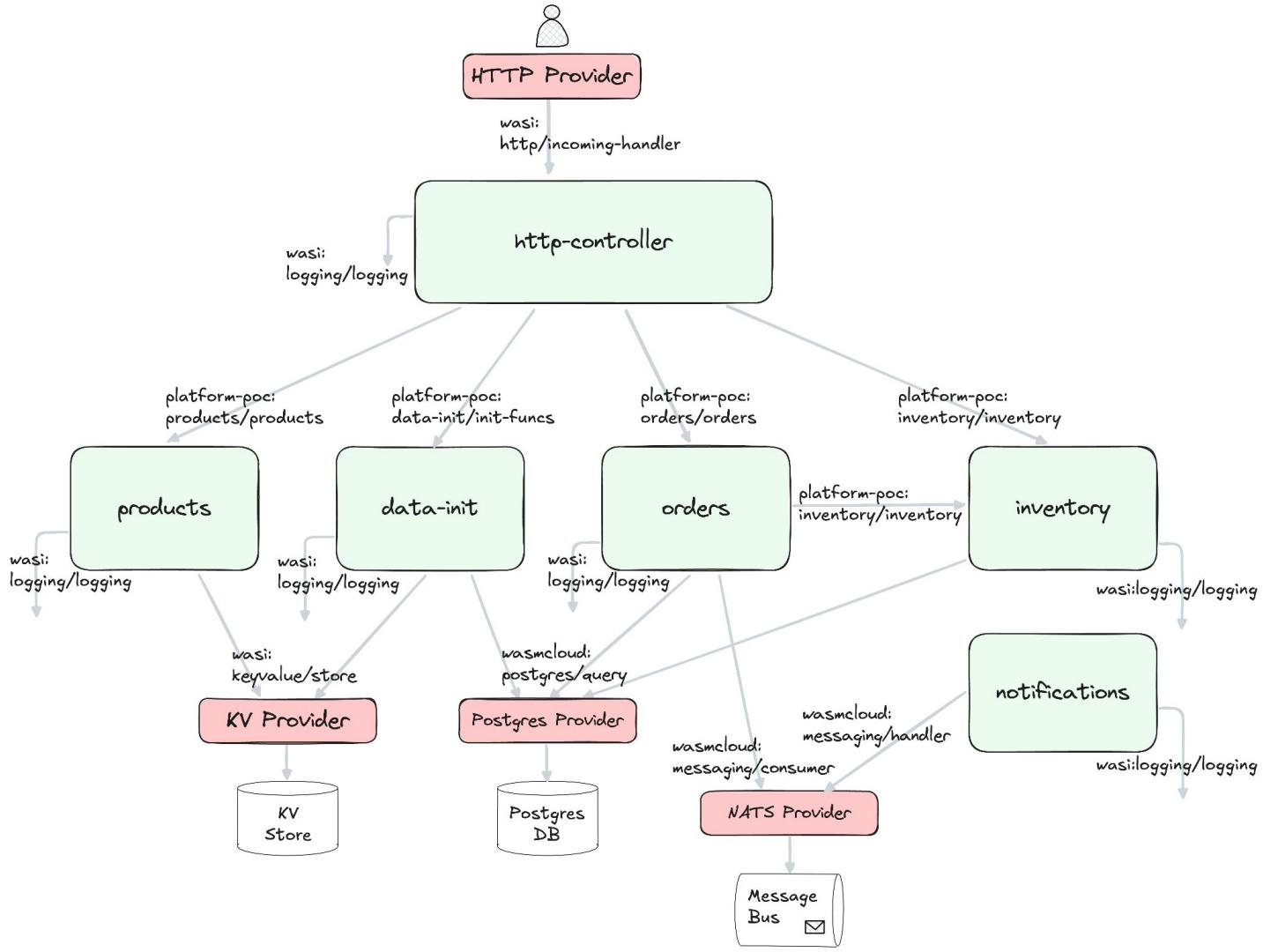
Example



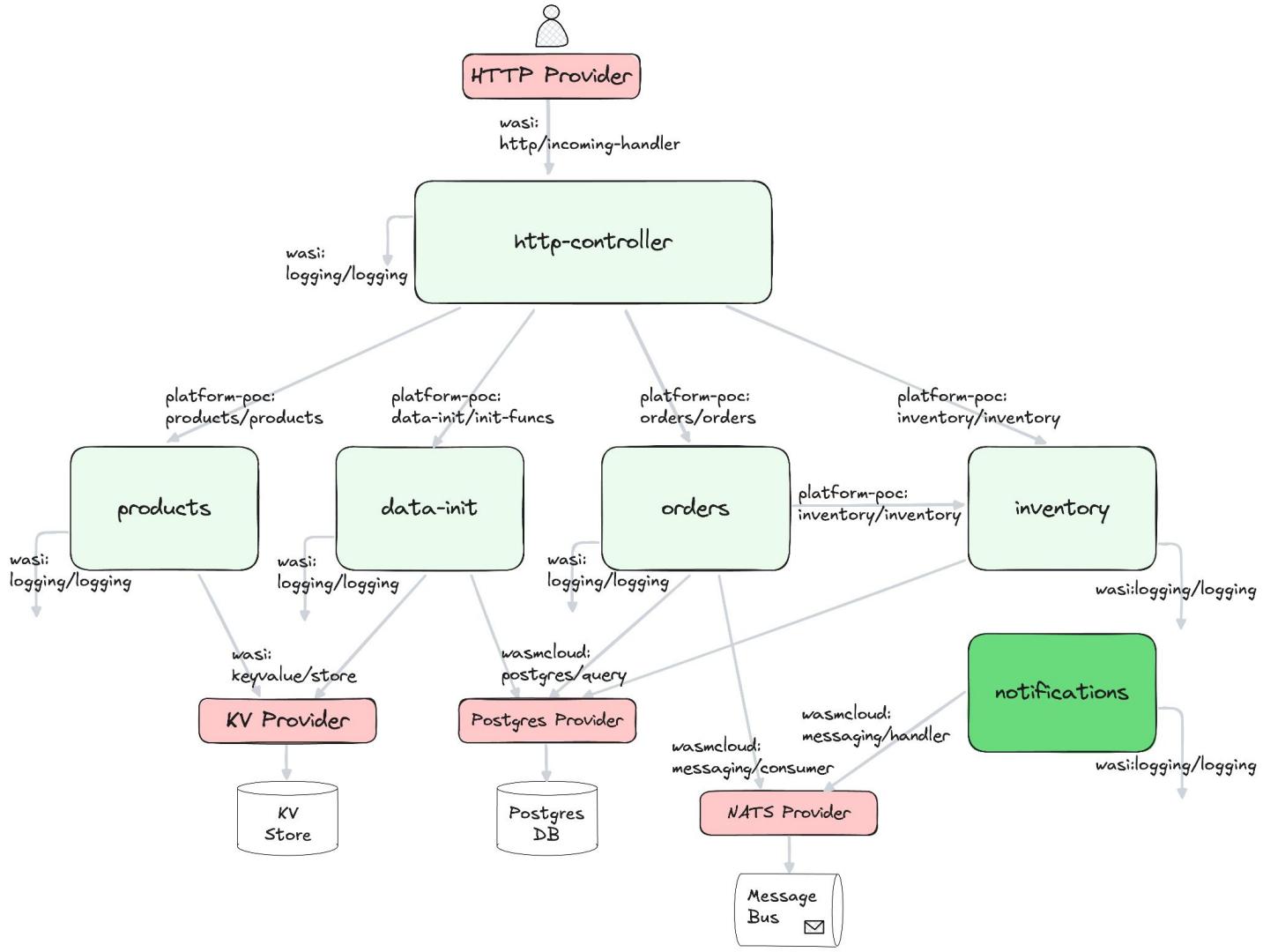
Example



Example



Example





Placing 10K orders

```
→ oha 'http://localhost:8080/orders' \
      -m POST \
      -d '[{"sku": "ENG-V8-500", "price": 1000, "quantity": 1}]' \
      -c 20 \
      -n 10000
```

Summary:

Success rate: 100.00%

Total: 4.3168 secs

Slowest: 0.0484 secs

Fastest: 0.0037 secs

Average: 0.0086 secs

Requests/sec: 2316.5356



Wasm binaries are small

About 1000 times smaller than their equivalent Java Spring Boot container image.

```
92k data_init_s.wasm
543k http_controller_s.wasm
70k inventory_service_s.wasm
89k notification_service_s.wasm
105k orders_service_s.wasm
100k products_service_s.wasm
```



Fermyon.com on Spin

“A fresh page load of Fermyon.com may result in starting, executing, and shutting down 30+ concurrent WebAssembly processes. Yet the result is a page that loads so fast that it scores a 99% on Google’s page speed ranking.”

“Not long ago, we had a very busy day. Our traffic jumped to ten times its normal number of page requests. At a few moments, we were seeing over 2,500 requests coming in at a time (90 concurrent users each fetching around 30 resources). In our model, we run three workers, each on a small-sized VM with 3 CPUs and 300Mb of memory per worker.”



Rust loves WebAssembly!

```
stuartharris .../hello-world master ? v1.85.1 11:14
→ cargo build --release --target wasm32-wasip1
Compiling hello-world v0.1.0 (/private/tmp/wasm/hello-world)
Finished `release` profile [optimized] target(s) in 0.63s
```

module
component

```
stuartharris .../hello-world master ? v1.85.1 11:16
→ cargo build --release --target wasm32-wasip2
Compiling hello-world v0.1.0 (/private/tmp/wasm/hello-world)
Finished `release` profile [optimized] target(s) in 0.47s
```

Safety and speed.

Increased productivity, really.

Shift left on quality.





Roadmap

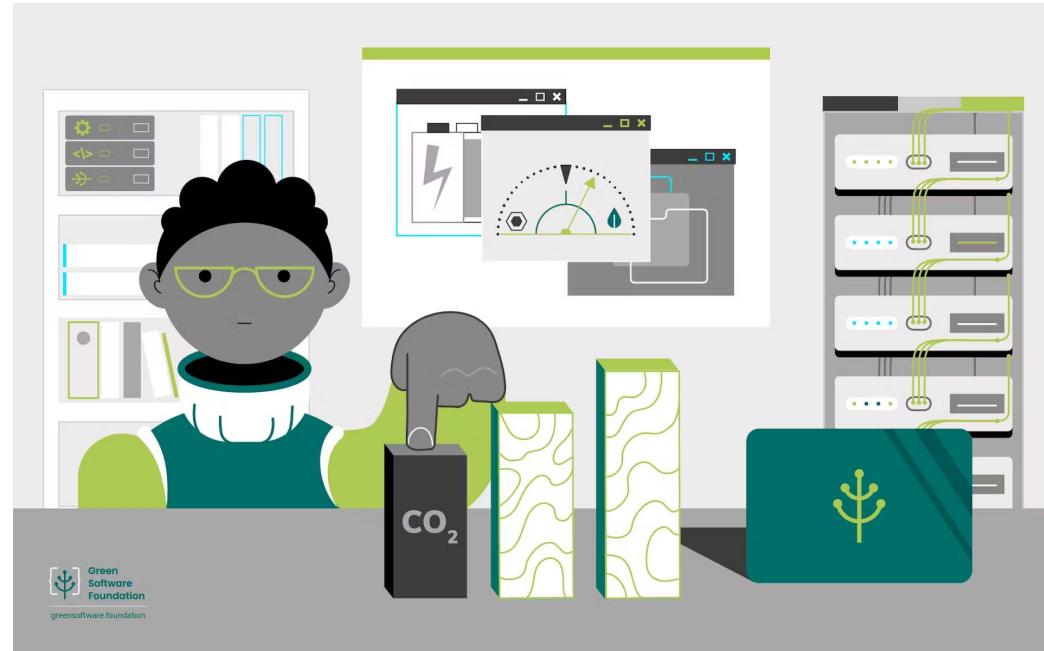
- Production ready — build greenfield scale-to-zero applications today
- Create a hybrid Kubernetes cluster using
 - ◆ wasmCloud Operator
 - ◆ SpinKube
- Gradually migrate, service by service, to new architecture
 - ◆ Remove all side effects, leaving just business logic

Let a decade of containers give way to a decade of WebAssembly components!



Let's build eco-friendly enterprise applications

- The Green Software Foundation [ref](#)
- The Principles of Sustainable Software Engineering [ref](#)



Thank you! 🙏



Stuart Harris
@stuartharris
stuart.harris@red-badger.com

[slides](#)



<https://github.com/rebadger/platform-poc>





Thank you

red-badger.com

hello@red-badger.com