FERMYON

Building new serverless primitives with Wasm

December 2024

Radu Matei

- CTO & co-founder of Fermyon
- Creator of the Spin project

- Maintainer and contributor to several **cloud-native** projects, at the intersection of **distributed systems**, **security**, and **developer experience**

- Loves bicycles, classical music, and bubble tea

@radu-matei.com on Bsky, https://radu-matei.com, **radu@fermyon.com**

Fermyon.com

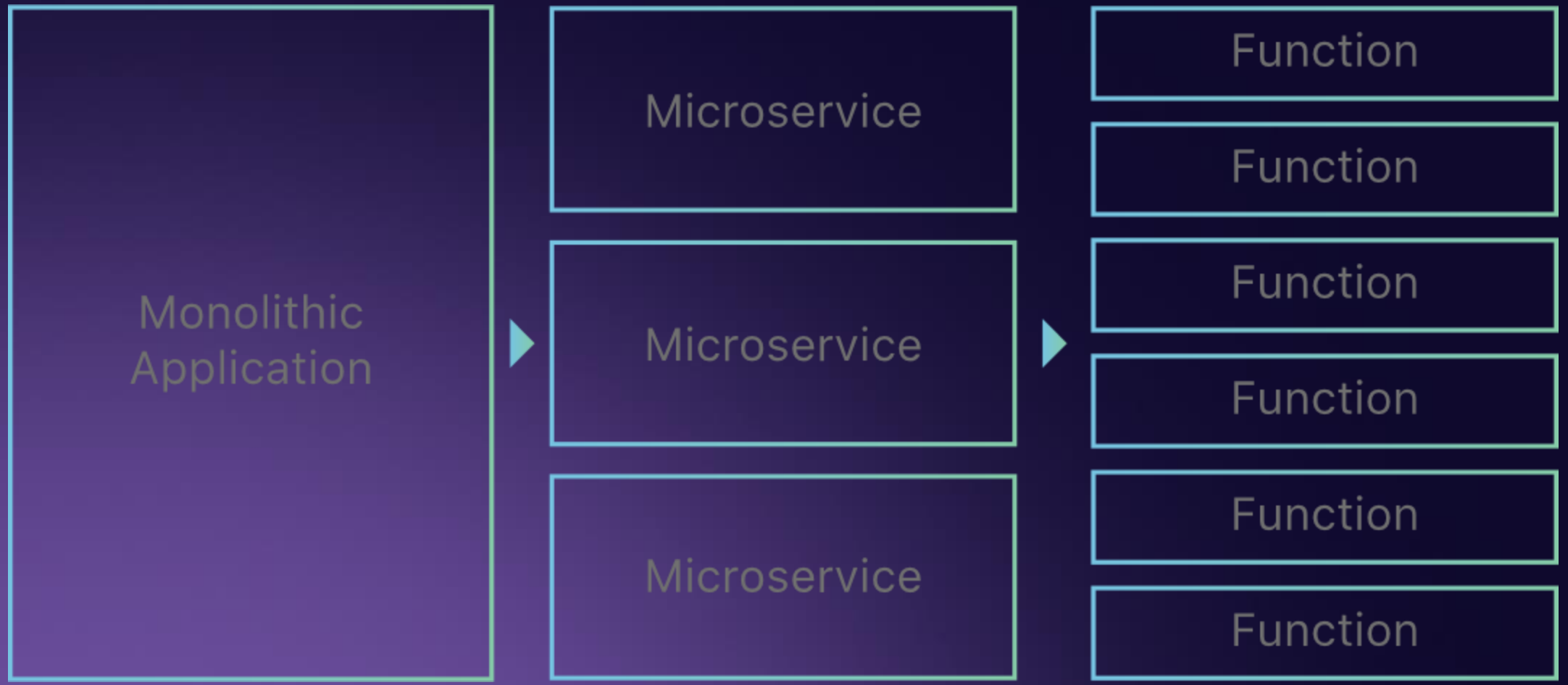
- started Fermyon in November '21 to work on Wasm
- built the Spin open source project
- Fermyon Cloud as the managed service for running serverless Wasm
- SpinKube and more cloud native tooling for Wasm
- core contributors and maintainers to ecosystem projects such as Wasmtime

Serverless: A coding pattern in which the developer does not write a server process.

The entry point is an event handler **function**.

FERMYON

From VMs to Functions



FERMYON

Developers love* writing Functions

Why FaaS:

- spending time writing application logic, not deploying servers
- not thinking about scaling up or down, and not being charged for idle resources**

*For the right use cases

**Unless on a platform that still uses VMs and "dedicated plans"

BUT...

- why do I have to rebuild my "serverless function" for ARM64?
- why do I experience 200+ms cold starts?
- once I build the function for this platform, can I run it elsewhere?

- the developer experience can be a pain*

*Will come back to developer experience

The Function is still executed in a microVM or container

- most popular FaaS platforms still use microVMs or containers to execute user functions.

- portability and binary size (and by extension, cold starts) we are experiencing with these platforms are stemmed in the unit of execution: **microVMs or containers**.

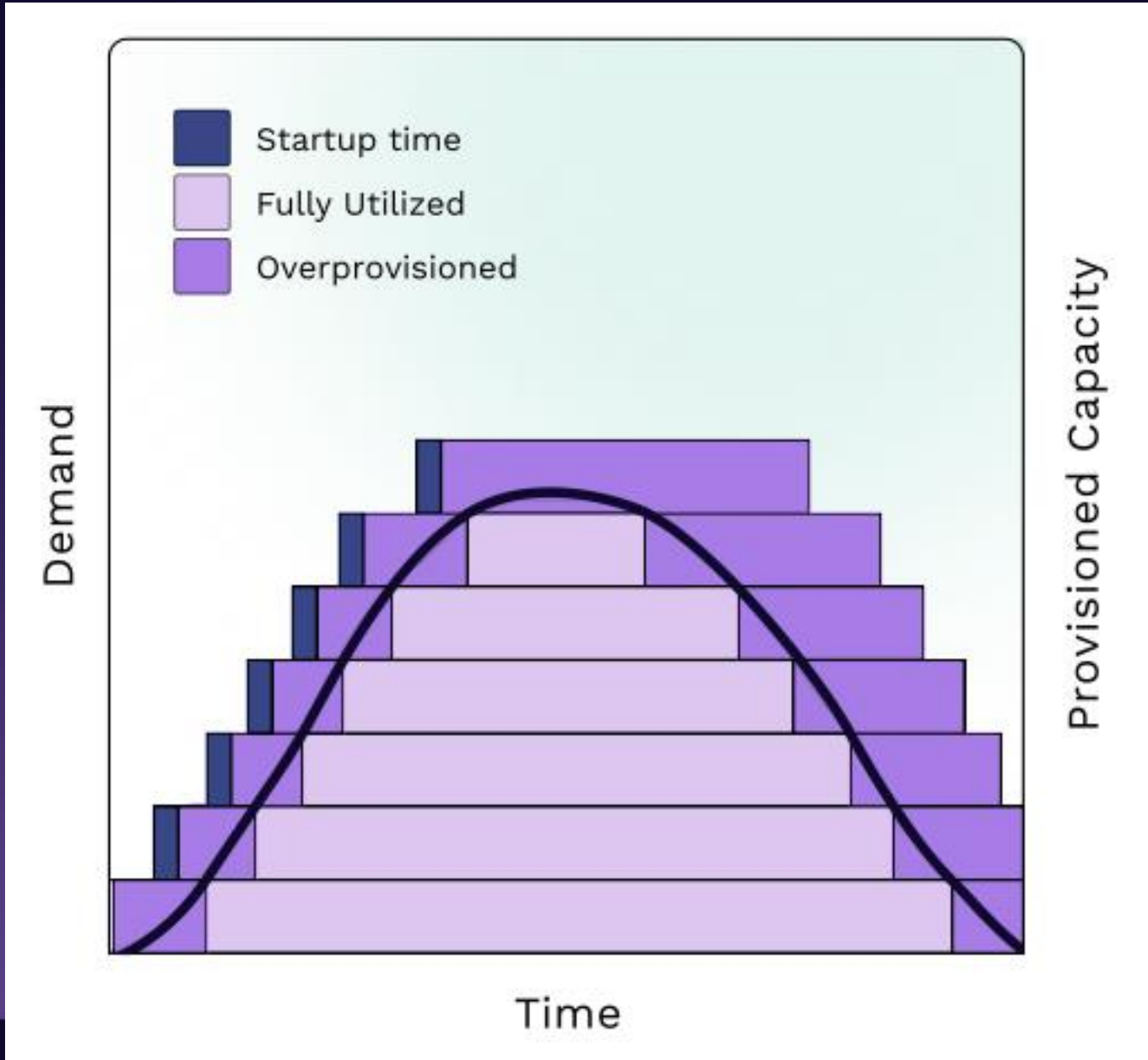
Efficiency for serverless

As a platform builder using microVMs or containers:

- **over**provision and pass the cost to end users

OR

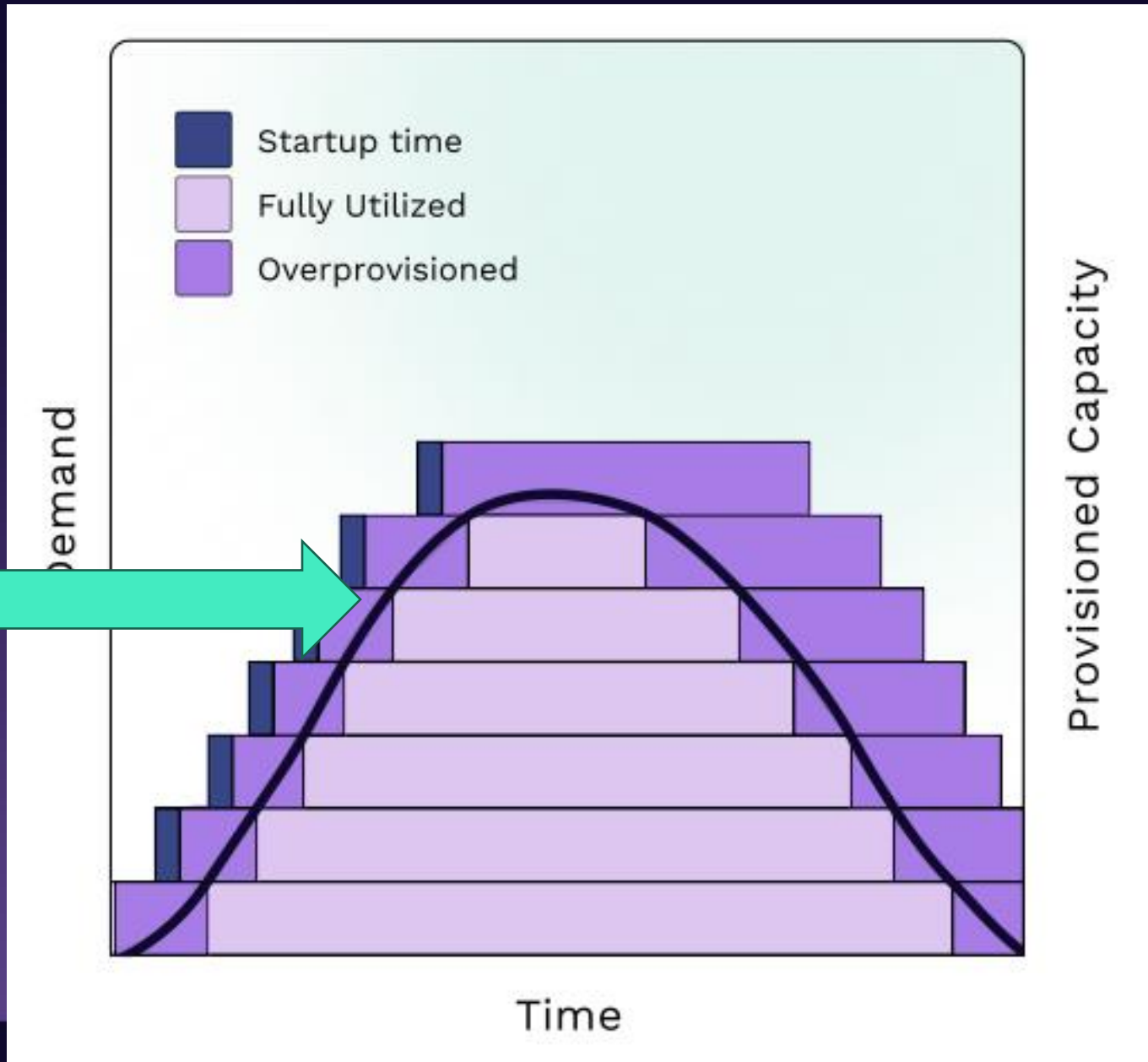
- **under**provision and pass the cold starts to end users



Efficiency for serverless

How do we make the shape of cloud compute better match the demand of this curve?

And by extension, pass BOTH cost savings AND fast startup to end users?



This is a big issue already for a single-region datacenter...

The problem becomes exponential if you need to schedule a function in hundreds of regions around the world and execute it as close as possible to an end user request, as fast as possible.

Do we really need microVMs and containers for FaaS?

From a developer perspective:

- if you need a full Linux OS in your function: YES.
- but for a growing number of use cases: NO (and platforms like Cloudflare Workers show us that with the right API surface, developers can achieve a lot without a full Linux OS available in their function)

What if we had another unit of deployment and execution for FaaS?

FERMYON

Enter Wasm

Or WebAssembly

What is WebAssembly (Wasm)?

- a target for compiling programs written in C/C++, Rust, Python, JavaScript, Kotlin, and more into a compact and portable intermediate binary format

- a runtime for executing these binaries in an isolation sandbox

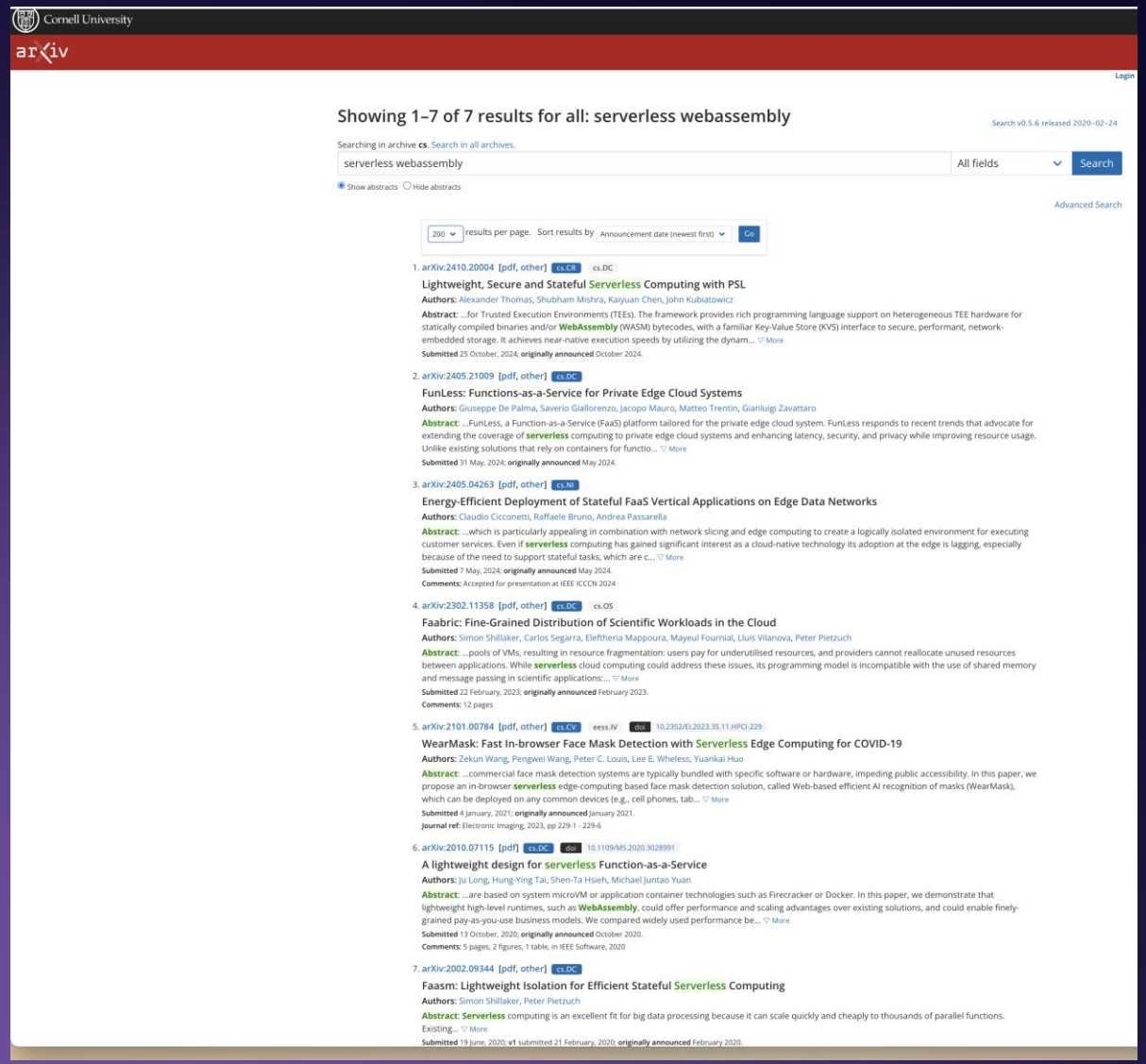
What is WebAssembly (Wasm)?

- originally built to port programs such as Photoshop or Google Earth to the browser, it evolved to run in other places, outside the browser

Why Wasm for serverless

- isolation sandbox (Wasm code does not have access to data outside its own linear memory or to APIs unless allowed by the host)
- cross platform and cross architecture portability (once compiled, a Wasm component can run on Linux, Windows, macOS, and on x86, ARM64, RISC-V)
- fast startup and near-native execution (Wasm components start in less than 1 millisecond if available on disk, and can be as little as a few MBs or less)
- can support a wide range of programming languages (write your functions in Rust, JavaScript, Python, Go, or more)

Academic research for Wasm and FaaS



Cornell University

arXiv

Login

Showing 1–7 of 7 results for all: serverless webassembly

Search v0.5.6 released 2020-02-24

Searching in archive **cs**. Search in all archives.

serverless webassembly | All fields | Search

Show abstracts | Hide abstracts

Advanced Search

200 results per page. Sort results by Announcement date (newest first) Go

1. arXiv:2410.20004 [pdf, other] cs.CR cs.DC

Lightweight, Secure and Stateful Serverless Computing with PSL

Authors: Alexander Thomas, Shubham Mishra, Kaiyuan Chen, John Kubiatowicz

Abstract: ...for Trusted Execution Environments (TEEs). The framework provides rich programming language support on heterogeneous TEE hardware for statically compiled binaries and/or **WebAssembly** (WASM) bytecodes, with a familiar Key-Value Store (KVS) interface to secure, performant, network-embedded storage. It achieves near-native execution speeds by utilizing the dynam... ▽ More

Submitted 25 October, 2024; **originally announced** October 2024.

2. arXiv:2405.21009 [pdf, other] cs.DC

FunLess: Functions-as-a-Service for Private Edge Cloud Systems

Authors: Giuseppe De Palma, Saverio Giallorenzo, Jacopo Mauro, Matteo Trentin, Gianluigi Zavattaro

Abstract: ...FunLess, a Function-as-a-Service (FaaS) platform tailored for the private edge cloud system. FunLess responds to recent trends that advocate for extending the coverage of **serverless** computing to private edge cloud systems and enhancing latency, security, and privacy while improving resource usage. Unlike existing solutions that rely on containers for functio... ▽ More

Submitted 31 May, 2024; **originally announced** May 2024.

3. arXiv:2405.04263 [pdf, other] cs.NI

Energy-Efficient Deployment of Stateful FaaS Vertical Applications on Edge Data Networks

Authors: Claudio Cicconetti, Raffaele Bruno, Andrea Passarella

Abstract: ...which is particularly appealing in combination with network slicing and edge computing to create a logically isolated environment for executing customer services. Even if **serverless** computing has gained significant interest as a cloud-native technology its adoption at the edge is lagging, especially because of the need to support stateful tasks, which are c... ▽ More

Submitted 7 May, 2024; **originally announced** May 2024.

Comments: Accepted for presentation at IEEE ICCCN 2024

4. arXiv:2302.11358 [pdf, other] cs.DC cs.OS

Faabric: Fine-Grained Distribution of Scientific Workloads in the Cloud

Authors: Simon Shillaker, Carlos Segarra, Eleftheria Mappoura, Mayeul Fournial, Lluis Vilanova, Peter Pietzuch

Abstract: ...pools of VMs, resulting in resource fragmentation: users pay for underutilised resources, and providers cannot reallocate unused resources between applications. While **serverless** cloud computing could address these issues, its programming model is incompatible with the use of shared memory and message passing in scientific applications:... ▽ More

Submitted 22 February, 2023; **originally announced** February 2023.

Comments: 12 pages

5. arXiv:2101.00784 [pdf, other] cs.CV eess.IV doi 10.2352/EI.2023.35.11.HPCI-229

WearMask: Fast In-browser Face Mask Detection with Serverless Edge Computing for COVID-19

Authors: Zekun Wang, Pengwei Wang, Peter C. Louis, Lee E. Wheless, Yuankai Huo

Abstract: ...commercial face mask detection systems are typically bundled with specific software or hardware, impeding public accessibility. In this paper, we propose an in-browser **serverless** edge-computing based face mask detection solution, called Web-based efficient AI recognition of masks (WearMask), which can be deployed on any common devices (e.g., cell phones, tab... ▽ More

Submitted 4 January, 2021; **originally announced** January 2021.

Journal ref: Electronic Imaging, 2023, pp 229-1 - 229-6

6. arXiv:2010.07115 [pdf] cs.DC doi 10.1109/MS.2020.3028991

A lightweight design for serverless Function-as-a-Service

Authors: Ju Long, Hung-Ying Tai, Shen-Ta Hsieh, Michael Juntao Yuan

Abstract: ...are based on system microVM or application container technologies such as Firecracker or Docker. In this paper, we demonstrate that lightweight high-level runtimes, such as **WebAssembly**, could offer performance and scaling advantages over existing solutions, and could enable finely-grained pay-as-you-use business models. We compared widely used performance be... ▽ More

Submitted 13 October, 2020; **originally announced** October 2020.

Comments: 5 pages, 2 figures, 1 table, in IEEE Software, 2020

7. arXiv:2002.09344 [pdf, other] cs.DC

Faasm: Lightweight Isolation for Efficient Stateful Serverless Computing

Authors: Simon Shillaker, Peter Pietzuch

Abstract: **Serverless** computing is an excellent fit for big data processing because it can scale quickly and cheaply to thousands of parallel functions. Existing... ▽ More

Submitted 19 June, 2020; **v1** submitted 21 February, 2020; **originally announced** February 2020.

SPIN

Single binary tool

One tool for development and runtime

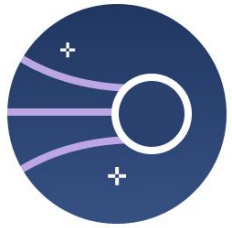
Many languages

Go, Rust, C#, Java, JavaScript, or any with Wasm+WASI support

Easy to get started

Use the template library, or provide your own and share components

No boilerplate required



Spin triggers get you right to the important part

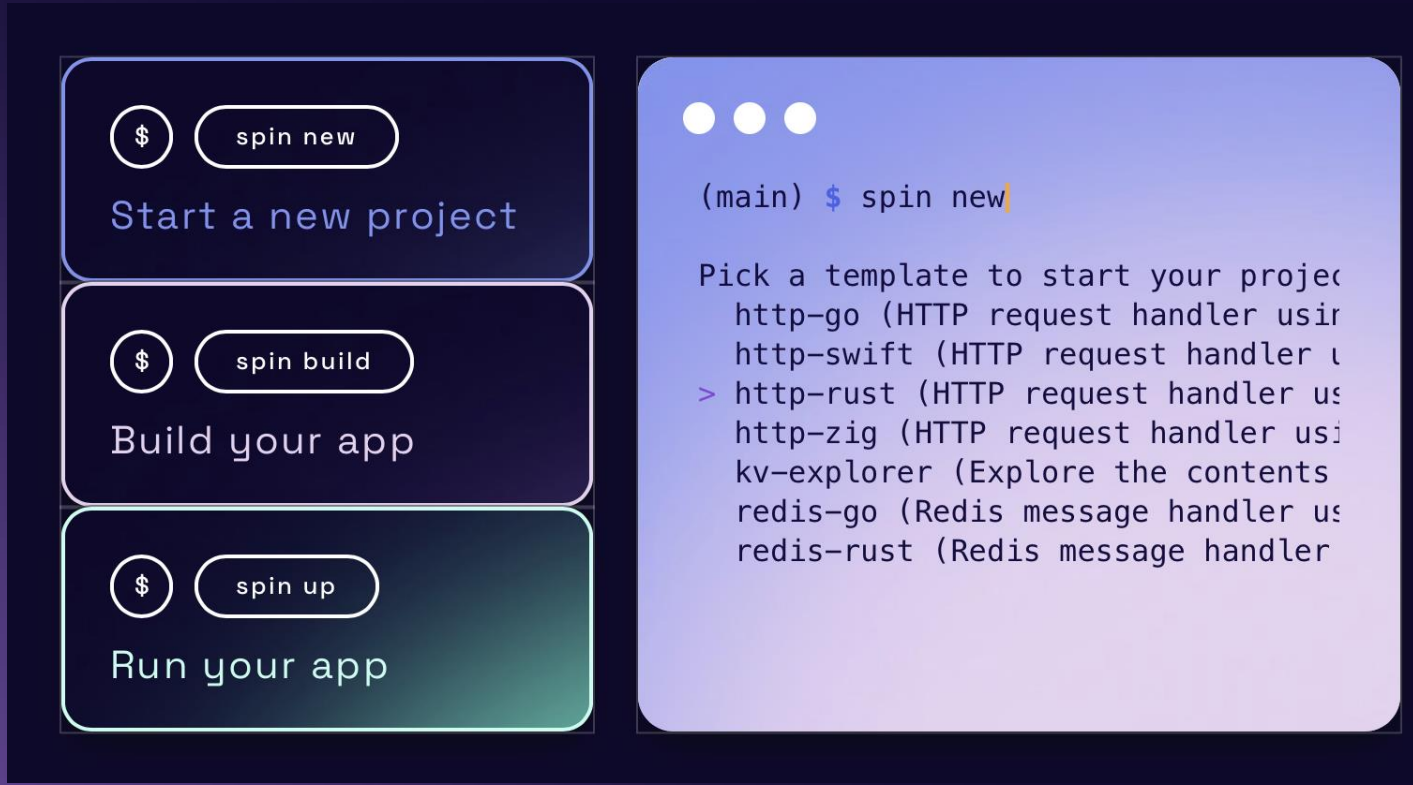
Work with data

Use Postgres, Redis and file storage to persist your data

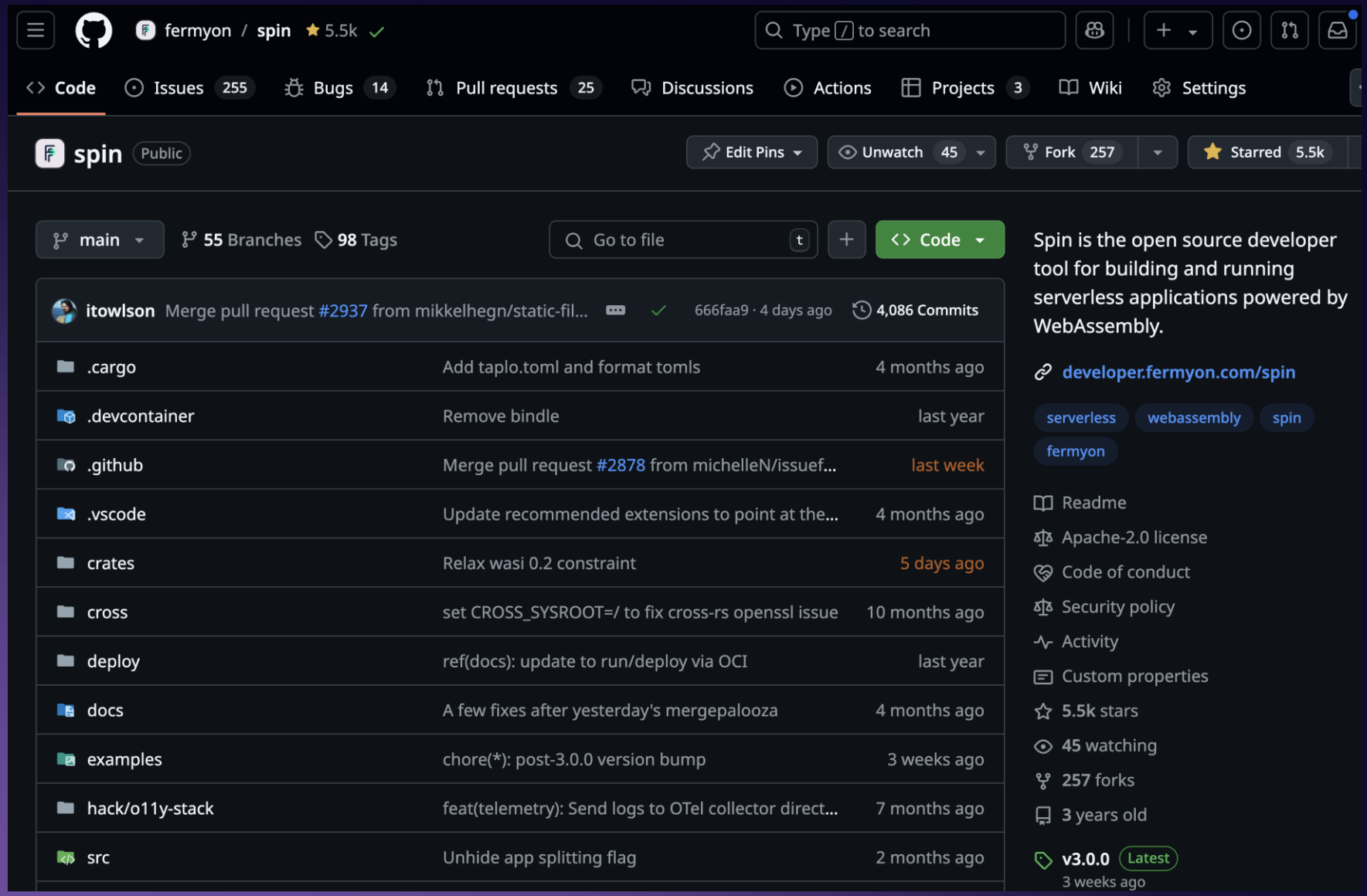
Spin

The open source tool for building WebAssembly serverless apps

With Spin, you can create a new serverless application (powered by one or more serverless functions) with just a few commands.



https://developer.fermyon.com/spin



https://github.com/fermyon/spin

FERMYON

FERMYON

Demo time

https://github.com/fermyon/spin

```
radu@asahi ☸ default in github.com/misc/hk
❯ bombardier localhost:3000 -c 512
Bombarding http://localhost:3000 for 10s using 512 connection(s)
[=========================================================]
Done!
Statistics        Avg      Stdev        Max
  Reqs/sec    138117.91   13199.46  161568.99
  Latency        3.70ms     1.36ms   105.99ms
  HTTP codes:
    1xx - 0, 2xx - 1381174, 3xx - 0, 4xx - 0, 5xx - 0
    others - 0
  Throughput:    21.33MB/s
```

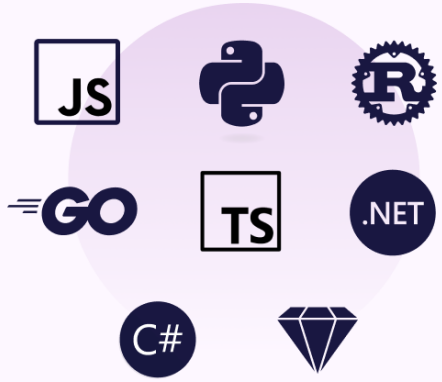
https://github.com/fermyon/spin

FERMYON

SPIN

The Developer Tool for Serverless WebAssembly

Bring Your Code



Supports many of the most popular programming languages

spin new

Serverless Made Simple


Docker Hub


GHCR


Sigstore


Redis


Key/Value Storage


HashiCorp Vault

Works with developer tools & registries.
Adds internal & external storage

spin build

spin up

Deploy & Run


Kuberenetes Cluster with Containerd

FERMYON CLOUD

spin cloud deploy

http://localhost:3000

What's new in Spin?

BUILD FULL-STACK APPLICATIONS

NEW

Serverless AI

Execute inferencing for LLMs directly from serverless apps.

BETA

SQLite Databases

Spin has a built-in database, which is always available - no Ops required.

Key/Value Store

Quickly persist data in your apps with Spin's in-built local KV store.

HTTP & Redis Triggers

Spin has a built-in HTTP web server and pub-sub Redis triggers, routing requests and messages to components.

Relational Database Storage

'Bring your own DB' support for MySQL and PostgreSQL, where you host and manage the database outside of Spin.

Variables & Secrets

Dynamic app variables mean a simpler experience for rotating secrets, updating API endpoints, and more.

Once you build your Spin app...

- run it with `spin up` on your own servers
- deploy it to Fermyon Cloud, soon run it globally
- deploy it to Kubernetes using the CNCF SpinKube project
- run it in other orchestrators such as Nomad or OpenShift
- run it directly in Docker Desktop using `docker run`

Wasm and the component model

- Spin is built on top Wasm components, defined using WIT (Wasm Interface Types)
- components are the core unit of execution, and encapsulate the function
- components can be statically analyzed and their capabilities allowed or declined

A Spin platform

```
package fermyon:spin@3.0.0;

/// The full world of a guest targeting an http-trigger
world http-trigger {
  include platform;
  export wasi:http/incoming-handler@0.2.0;
}

/// The imports needed for a guest to run on a Spin host
world platform {
  include fermyon:spin/platform@2.0.0;
  include wasi:keyvalue/imports@0.2.0-draft2;
  import spin:postgres/postgres@3.0.0;
  import wasi:config/store@0.2.0-draft-2024-09-27;
}
```

A Spin function

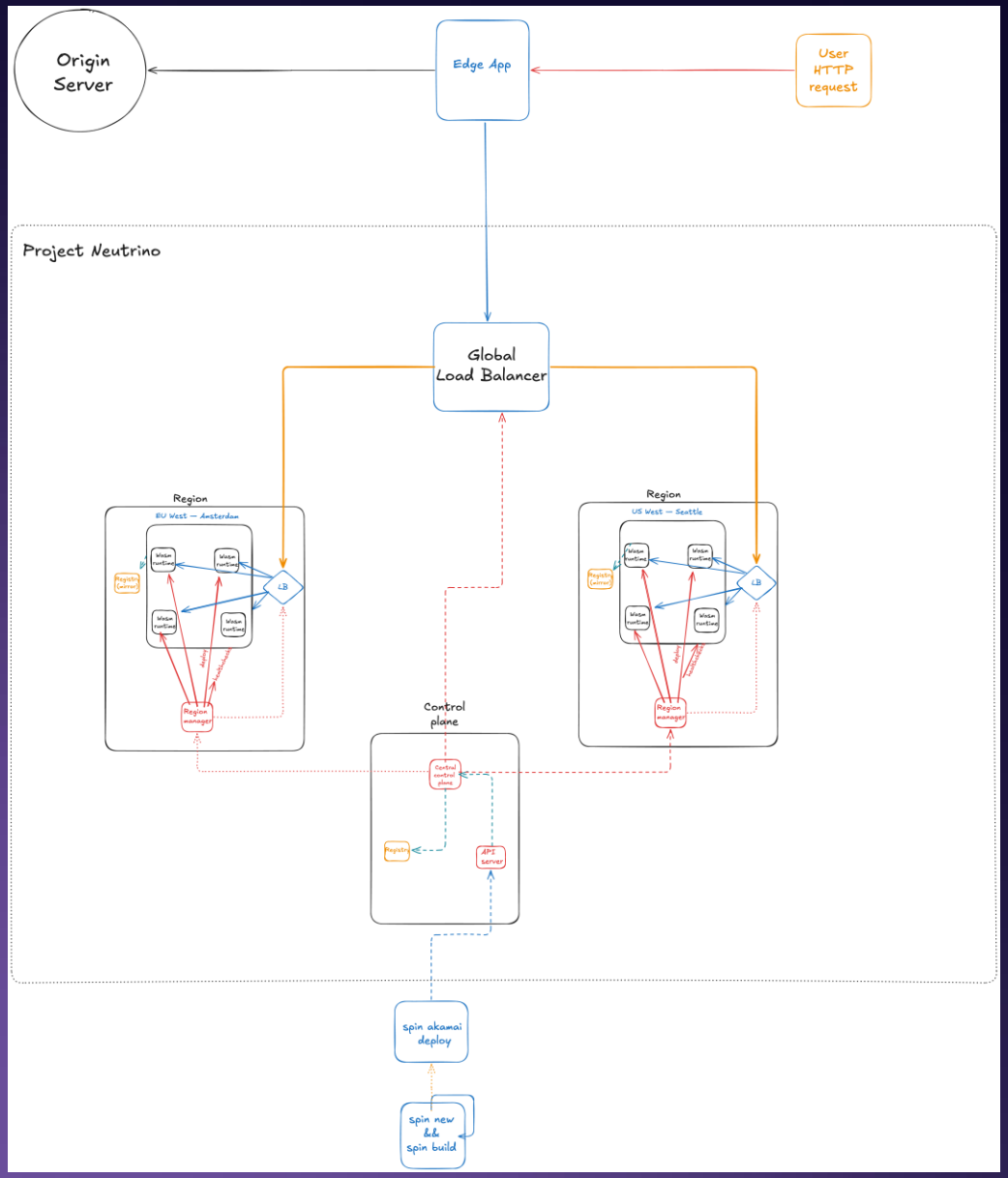
```
package root:root;

world root {
  import wasi:io/...@0.2.0;
  import wasi:clocks/...@0.2.0;
  import wasi:http/...@0.2.0;
  import wasi:config/...@0.2.0
  import wasi:cli/...@0.2.0;
  import wasi:filesystem/...@0.2.0;
  import wasi:sockets/...@0.2.0;
  import wasi:random/...@0.2.0;
  import wasi:keyvalue/...@0.2.0-draft2;

  import fermyon:spin/...@3.0.0;

  export wasi:http/incoming-handler@0.2.0;
}
```

What about multi-region?



What about isolation?

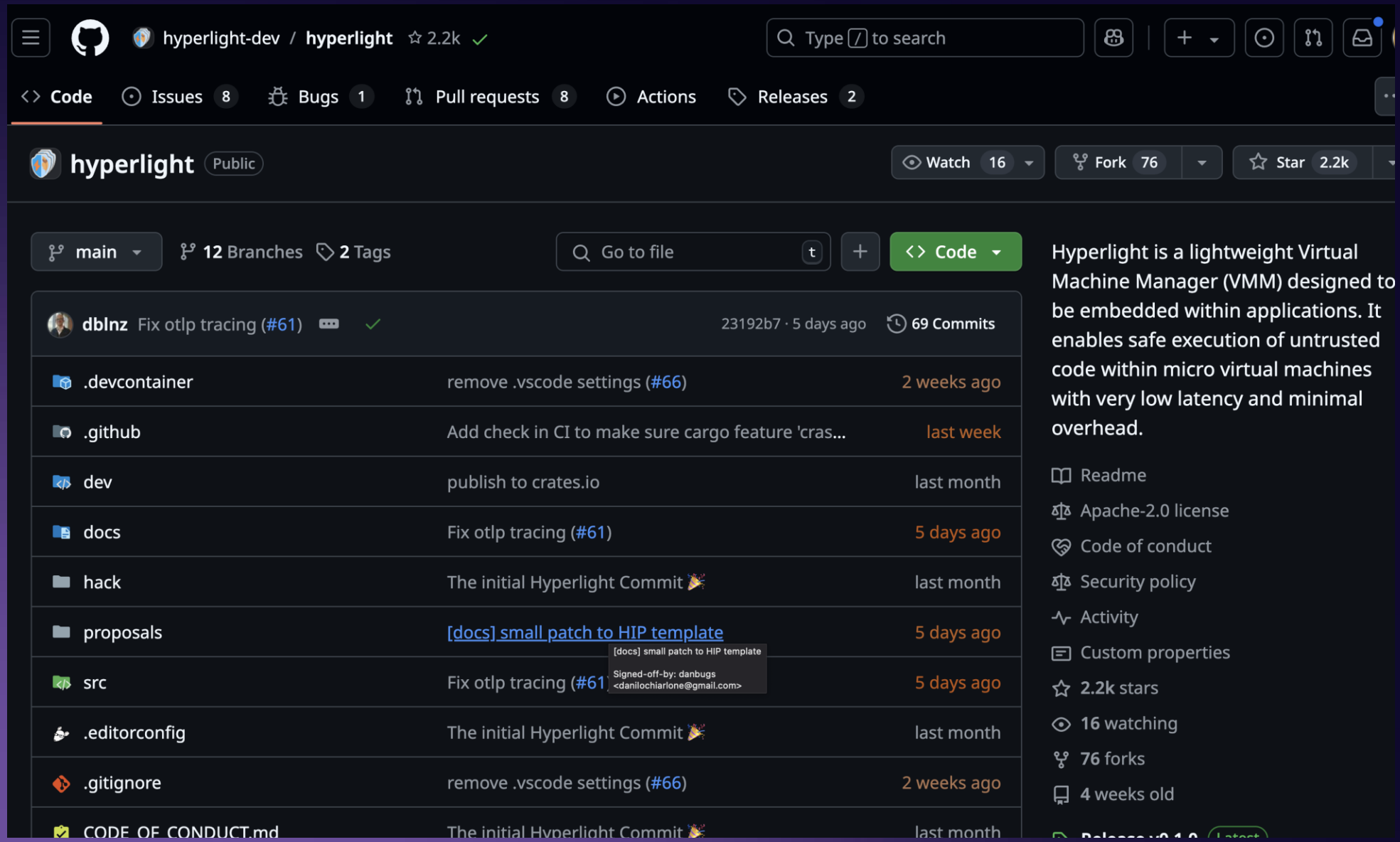
Security and Correctness in Wasmtime

Sep 13, 2022 Nick Fitzgerald

The essence of software engineering is making trade-offs, and sometimes engineers even trade away security for other priorities. When it comes to running untrusted code from unknown sources, however, exceptionally strong security is simply the bar to clear for serious participation: consider the extraordinary efforts that Web browser and hypervisor maintainers take to preserve their systems' integrity. WebAssembly runtimes also run untrusted code from untrusted sources, and therefore such efforts are also a hard requirement for WebAssembly runtimes.

bytecodealliance.org/articles/security-and-correctness-in-wasmtime

What about stronger isolation?



github.com/hyperlight-dev/hyperlight

Executing Wasm Functions at scale

- we can control the amount of memory, CPU time, network bandwidth a Function can use, and terminate at any given point in time

- we can centrally compile Wasm ahead-of-time to the specific OS/arch of data plane nodes and distribute highly optimized machine code

Executing Wasm Functions at scale #2

- we optimistically schedule thousands* of Functions on any given node in Fermyon Cloud (use cases favor short, bursty executions compared to long-running functions)

- with a mix of Linux x86 and ARM64 nodes, depending on efficiency, region availability

- we suspend in memory any function waiting for I/O such as file access or outbound network calls

Executing Wasm Functions at scale #3

- for high core count machines, we can pin the Wasm runtime on a subset of cores to minimize IPIs

Check out Spin and SpinKube.dev!

Get started with Spin and SpinKube!

Spin: https://github.com/fermyon/spin

SpinKube: https://github.com/spinkube

FERMYON