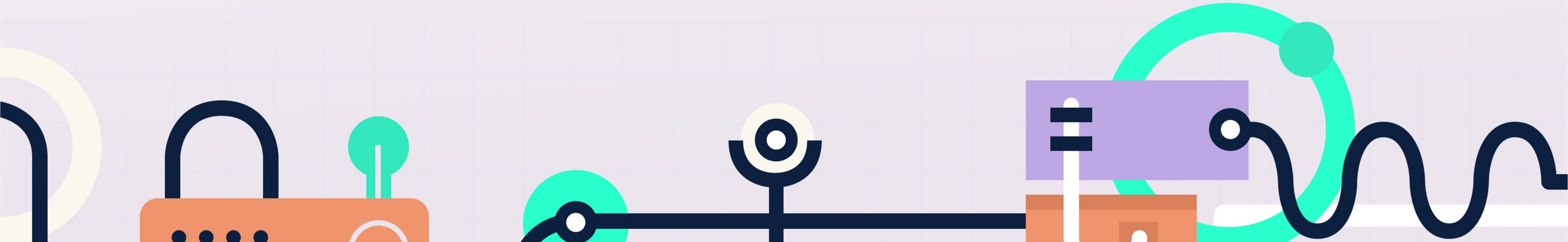
FERMYON

Microservices with Rust, WebAssembly, and the component model



Radu Matei

I am the co-founder and CTO of Fermyon, passionate about WebAssembly, distributed systems, and artificial intelligence.

When I am not around computers, I enjoy classical music, cycling, and bubble tea.

FERMYON

FERMYON

A few months ago, we launched Spin, Fermyon's first major open source project.

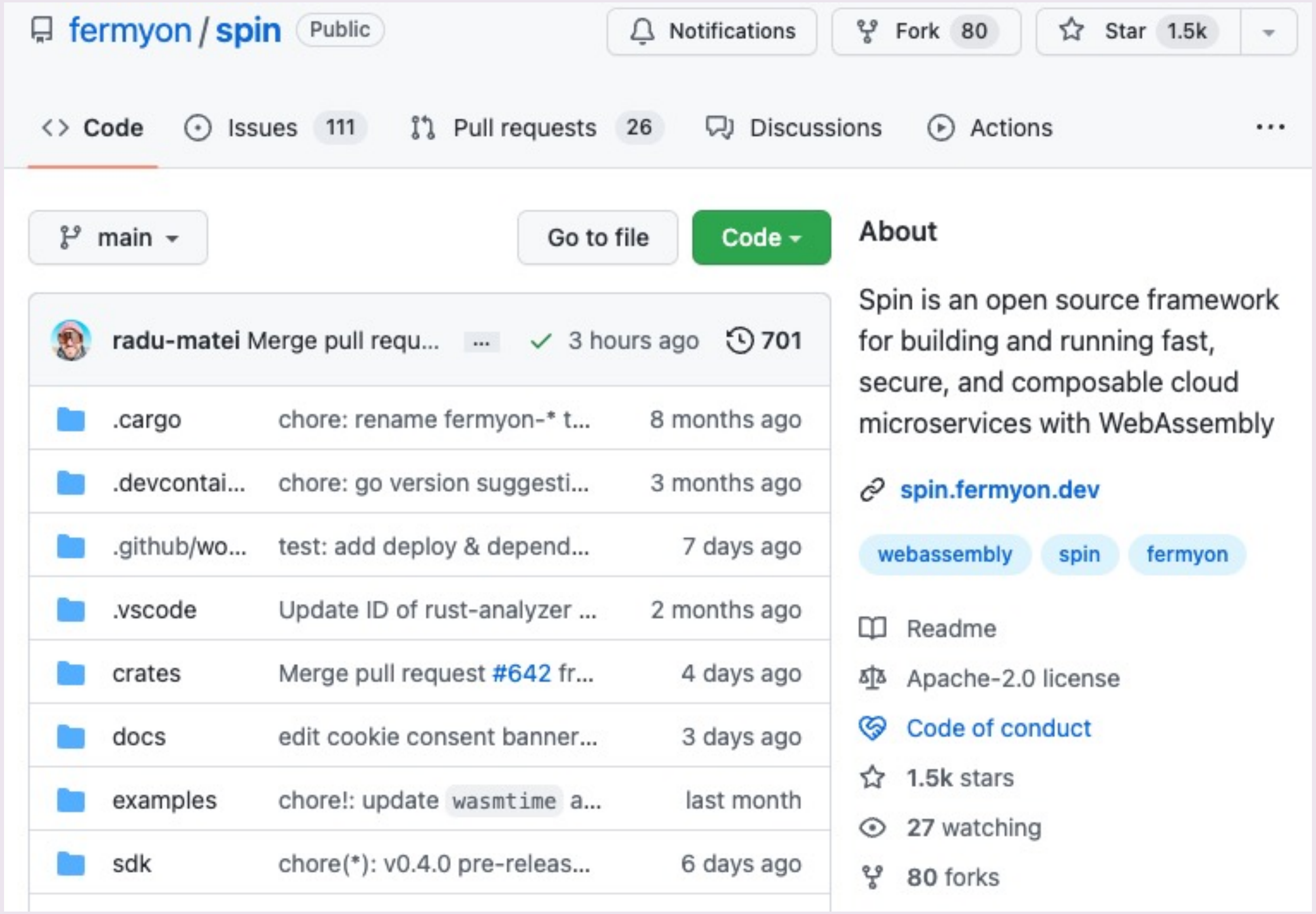
FERMYON

FERMYON

https://spin.fermyon.dev

FERMYON

github.com/fermyon/spin



fermyon / spin Public

Notifications | Fork 80 | Star 1.5k

Code | Issues 111 | Pull requests 26 | Discussions | Actions | ···

main | Go to file | Code

radu-matei Merge pull requ... ··· ✓ 3 hours ago 701

.cargo	chore: rename fermyon-* t...	8 months ago
.devcontai...	chore: go version suggesti...	3 months ago
.github/wo...	test: add deploy & depend...	7 days ago
.vscode	Update ID of rust-analyzer ...	2 months ago
crates	Merge pull request #642 fr...	4 days ago
docs	edit cookie consent banner...	3 days ago
examples	chore!: update `wasmtime` a...	last month
sdk	chore(*): v0.4.0 pre-releas...	6 days ago

About

Spin is an open source framework for building and running fast, secure, and composable cloud microservices with WebAssembly

spin.fermyon.dev

webassembly | spin | fermyon

Readme

Apache-2.0 license

Code of conduct

1.5k stars

27 watching

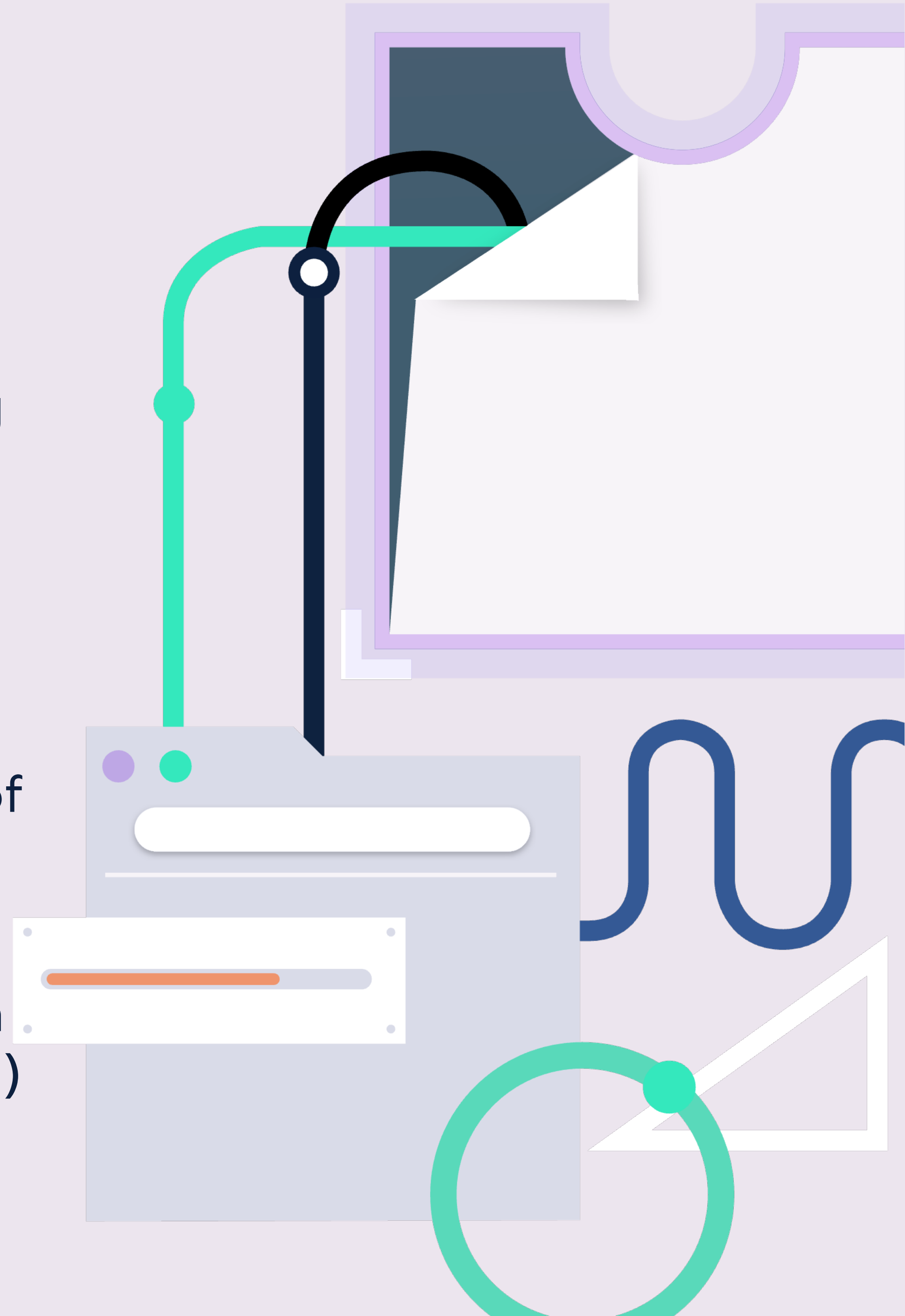
80 forks

FERMYON

Fermyon Spin

https://github.com/fermyon/spin

- an open source framework for building and running fast, secure, and composable microservices with WebAssembly
- with Spin, we want to make it easier to get started with WebAssembly on the server, so you can take advantage of the portability, security, and speed of Wasm.
- written in Rust, lets you write microservices written in Rust (and other languages that compile to WASI)



Your first Spin application in Rust

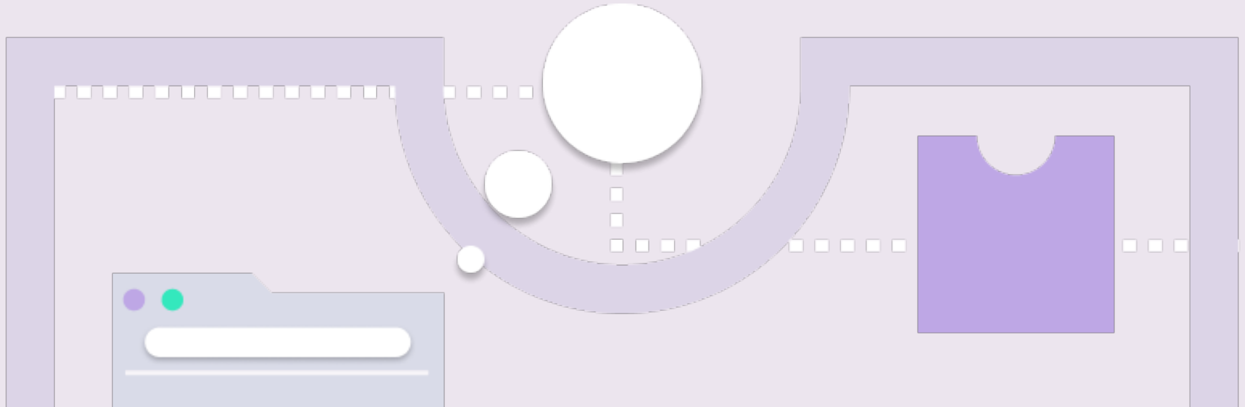
```
# create a new application based on the Rust HTTP template
$ spin new http-rust hello-world
# build it and start the application locally
$ spin build --up
```

```rust
#[spin_sdk::http_component]
fn hello_world(req: Request) -> Result<Response> {
   Ok(http::Response::builder()
       .status(200)
       .body(Some("Hello, RustAU!".into()))?)
}
```

FERMYON

“WebAssembly… defines a portable, size- and load-time-efficient format and execution model specifically designed to serve as *a compilation target* for the Web.”

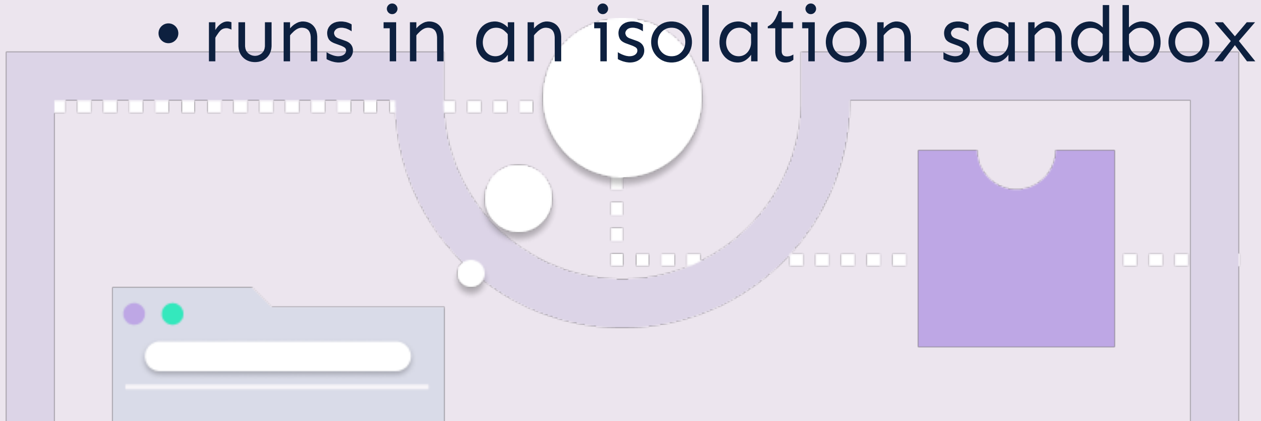
- Luke Wagner



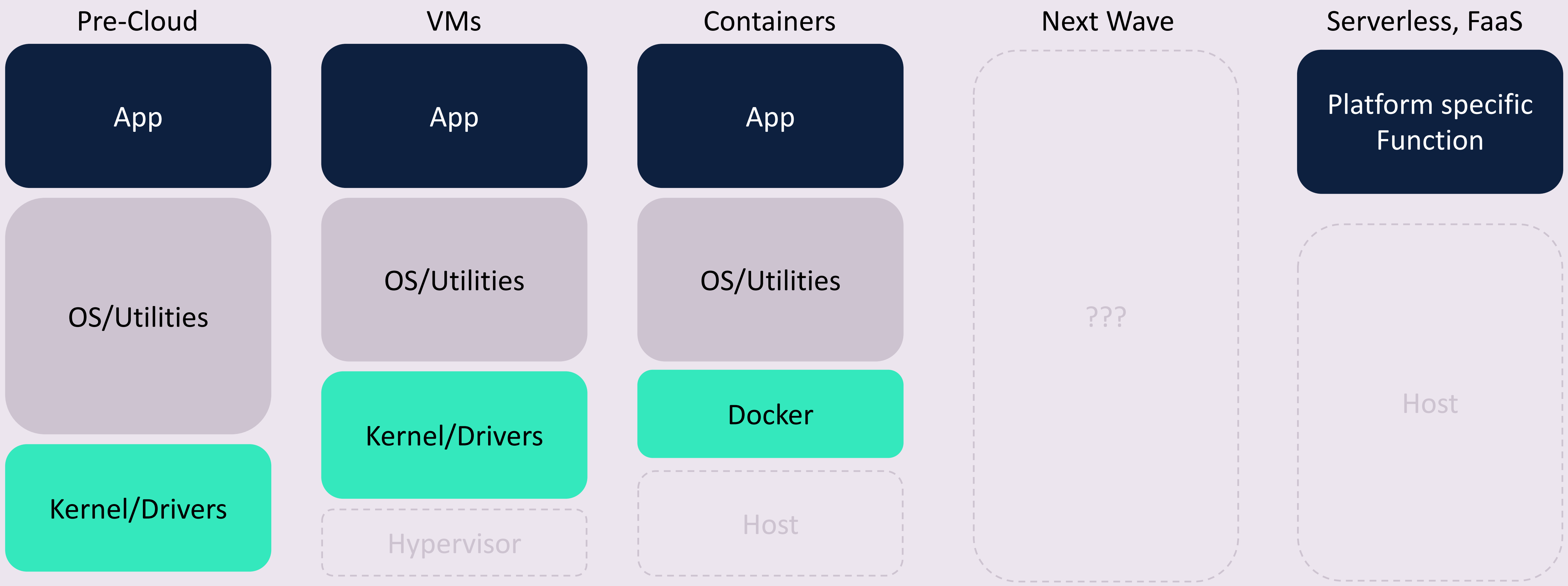
https://blog.mozilla.org/luke/2015/06/17/webassembly/

WebAssembly

- cross platform, cross architecture
- compact binary size
- fast startup times
- runs in an isolation sandbox

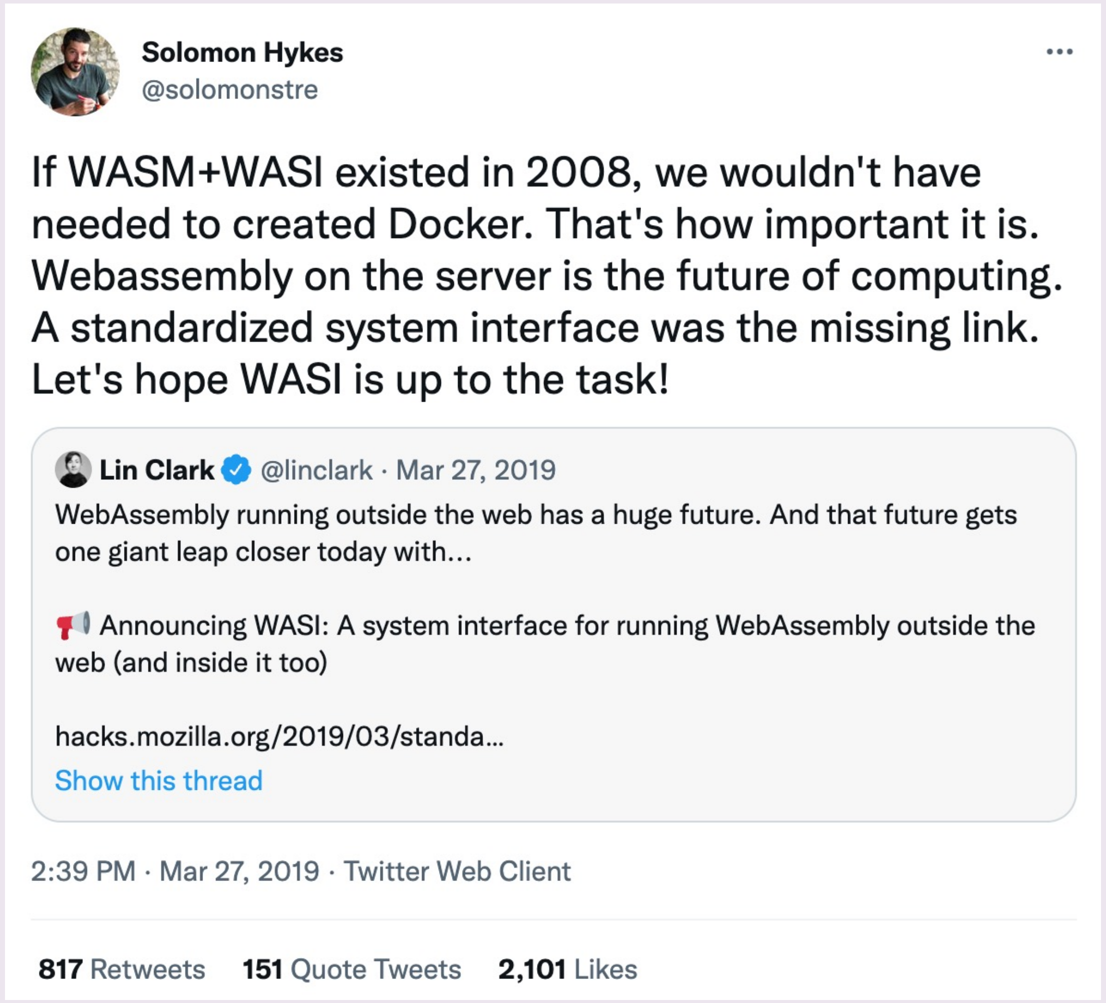


Cloud Computing



Wasm: Good for the browser, great for the cloud

- Cross platform, cross architecture, multi-language
- Small binary sizes
- Security sandbox
- Fast startup times, near-native speed
- Scalable from zero to N (and back again)



Solomon Hykes
@solomonstre

If WASM+WASI existed in 2008, we wouldn't have needed to created Docker. That's how important it is. Webassembly on the server is the future of computing. A standardized system interface was the missing link. Let's hope WASI is up to the task!

Lin Clark @linclark · Mar 27, 2019
WebAssembly running outside the web has a huge future. And that future gets one giant leap closer today with...

📢 Announcing WASI: A system interface for running WebAssembly outside the web (and inside it too)

hacks.mozilla.org/2019/03/standa...
Show this thread

2:39 PM · Mar 27, 2019 · Twitter Web Client

817 Retweets **151** Quote Tweets **2,101** Likes

FERMYON

Demo Time!

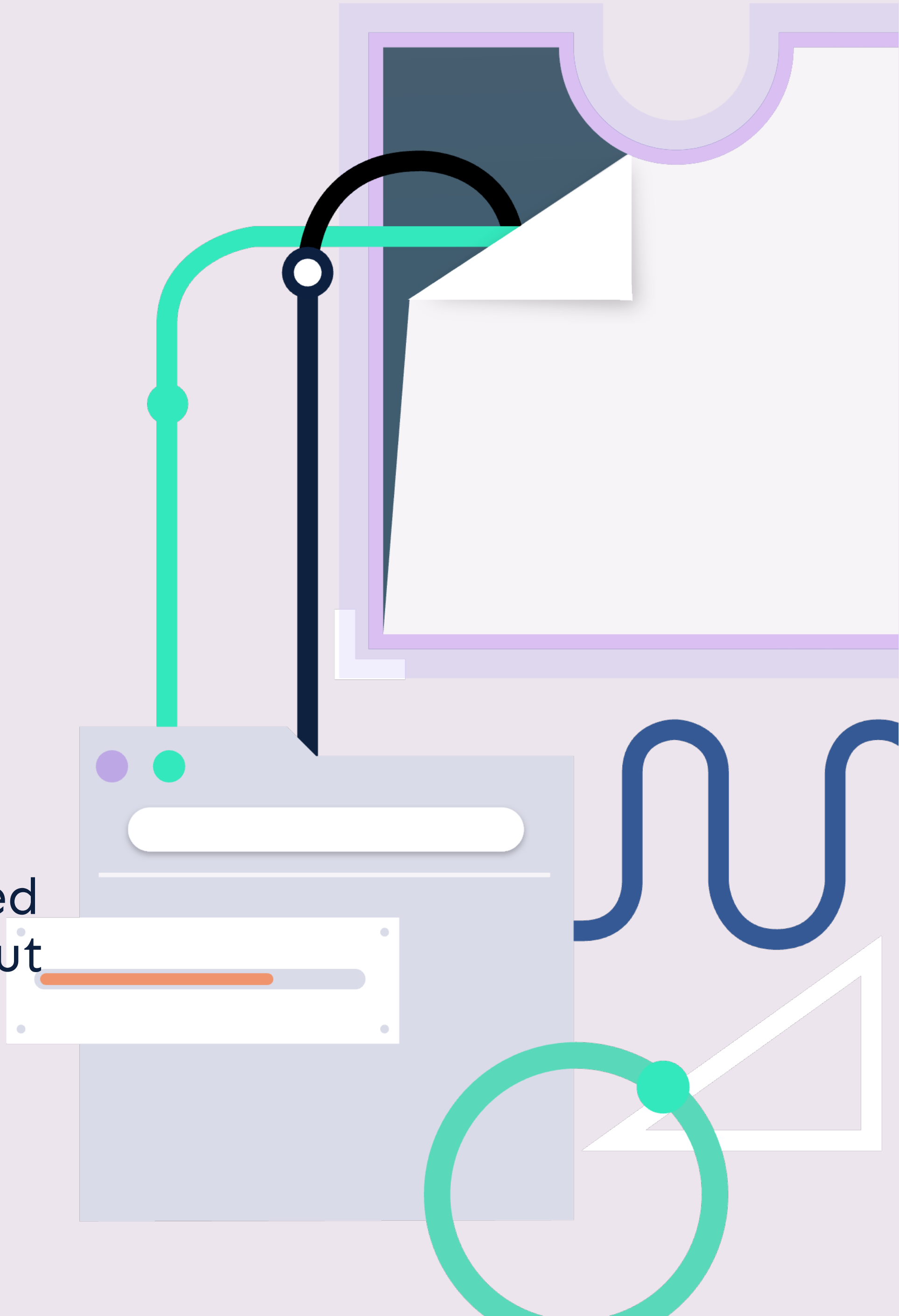
Building and running your first Spin app

FERMYON

How does Spin work?

https://github.com/fermyon/spin

- built on Bytecode Alliance tooling such as the WebAssembly component model and Wasmtime
- you build your application into a WebAssembly component, and publish it to a registry
- for each new request, Spin will create a new, isolated WebAssembly instance, handle the request, then shut it down



Not just HTTP!

https://github.com/fermyon/spin

- Spin also has built-in support for pub/sub with Redis
- your component gets instantiated and invoked for each new message on a Redis channel
- you can extend the Spin application model to build your own triggers! It is open source, built on THE WebAssembly standard.
- new models and triggers we know people want to build with Spin — WebSockets, MQTT, timer based triggers, and more!

Not just Rust!

https://github.com/fermyon/spin

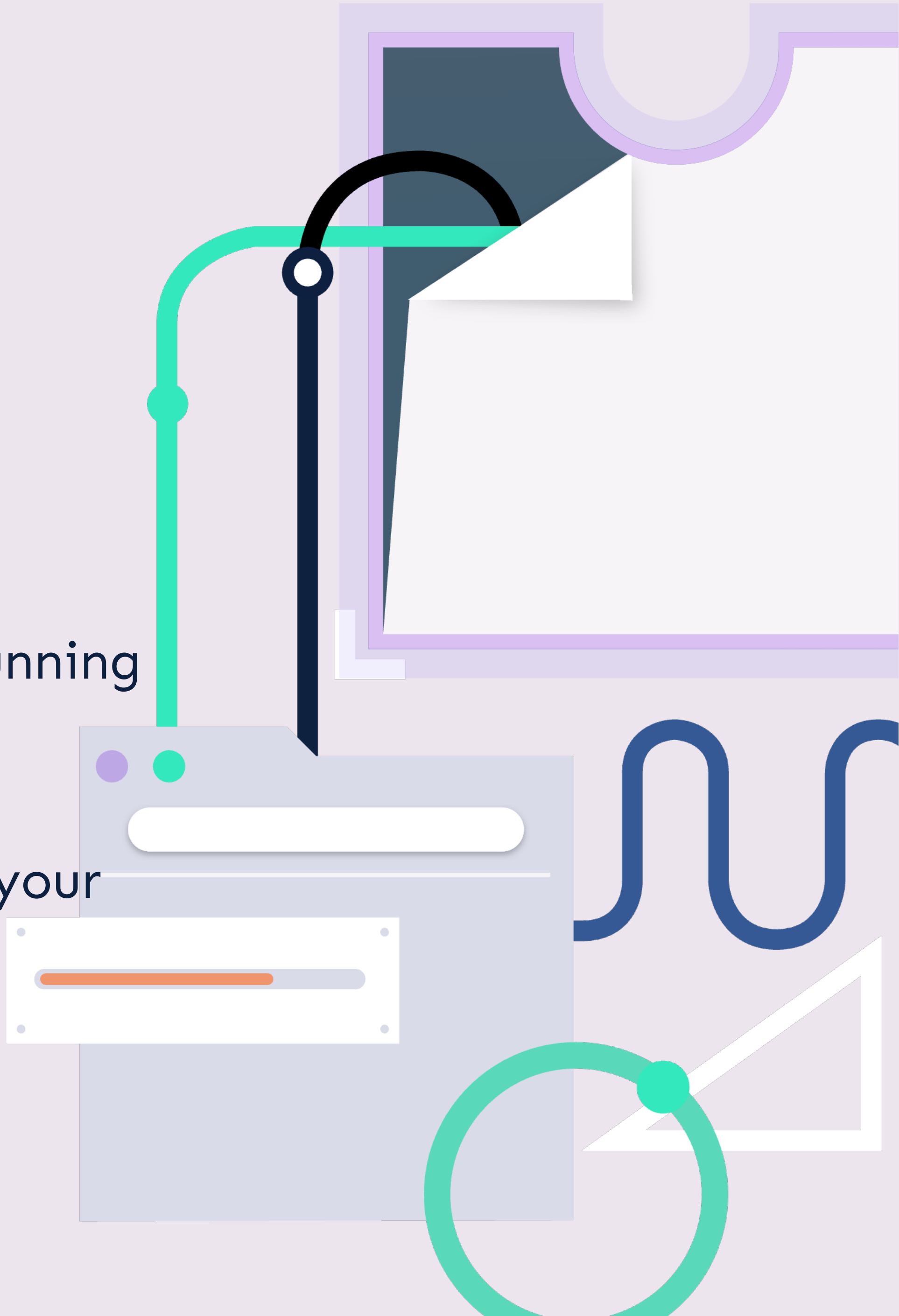
- Spin is written in Rust, and Rust has EXCELLENT support for WebAssembly
- but Spin lets you run components written in any language that compiles to WASI (the WebAssembly System Interface)!
- Spin has language SDKs for Rust and Go, and we are planning on adding more languages!

Not just a developer tool!

https://fermyon.dev

```
# create a new application based on the Rust HTTP template
$ spin new http-rust hello-world
# deploy your application to the Fermyon platform!
$ spin deploy
```

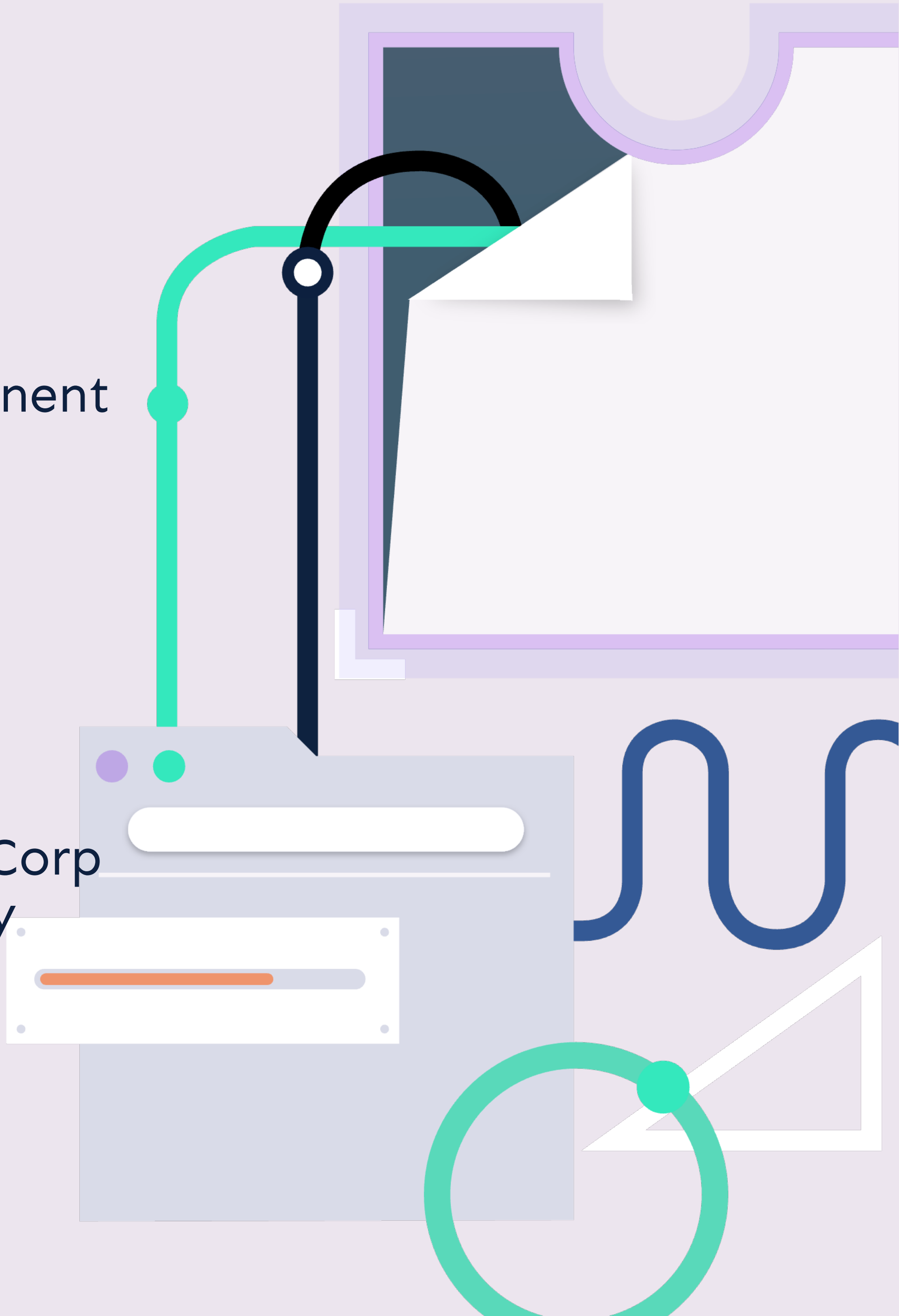
- we recently released the open source platform for running your Spin applications!
- `spin deploy`, and your application is now running in your infrastructure
- what is next?



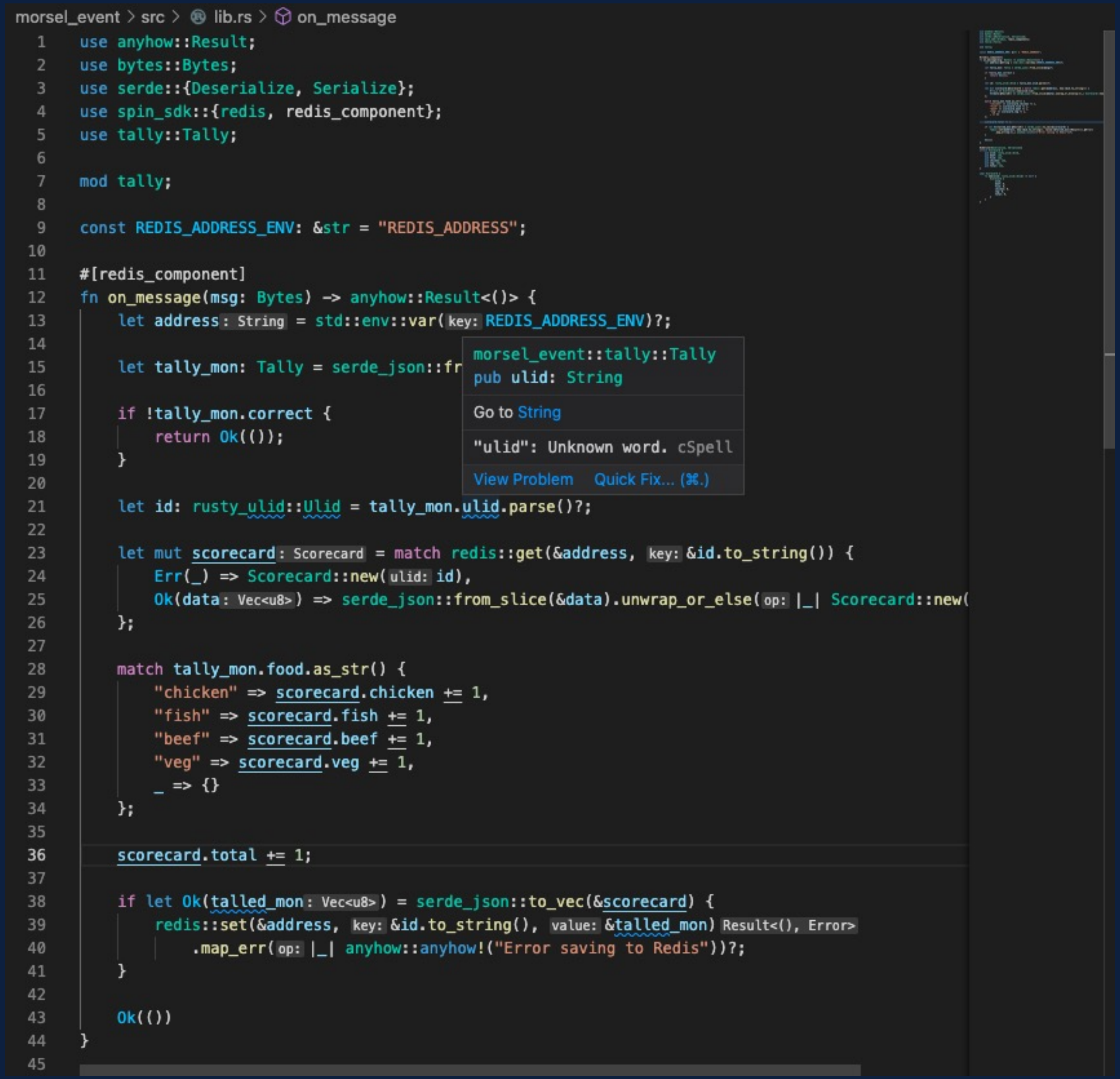
Not just WebAssembly!

https://fermyon.dev

- we wholeheartedly believe the WebAssembly component model is the future of distributed computing!
- but containers will still be around for a long time...
- the open source Fermyon Platform is built on HashiCorp Nomad, so you can run containers and WebAssembly microservices side by side



Finicky Whiskers is Wasm + Containers


morsel_event > src > lib.rs > on_message

```rust
use anyhow::Result;
use bytes::Bytes;
use serde::{Deserialize, Serialize};
use spin_sdk::{redis, redis_component};
use tally::Tally;

mod tally;

const REDIS_ADDRESS_ENV: &str = "REDIS_ADDRESS";

#[redis_component]
fn on_message(msg: Bytes) -> anyhow::Result<()> {
    let address: String = std::env::var(key: REDIS_ADDRESS_ENV)?;

    let tally_mon: Tally = serde_json::fr

    if !tally_mon.correct {
        return Ok(());
    }

    let id: rusty_ulid::Ulid = tally_mon.ulid.parse()?;

    let mut scorecard: Scorecard = match redis::get(&address, key: &id.to_string()) {
        Err(_) => Scorecard::new(ulid: id),
        Ok(data: Vec<u8>) => serde_json::from_slice(&data).unwrap_or_else(op: |_| Scorecard::new(
    };

    match tally_mon.food.as_str() {
        "chicken" => scorecard.chicken += 1,
        "fish" => scorecard.fish += 1,
        "beef" => scorecard.beef += 1,
        "veg" => scorecard.veg += 1,
        _ => {}
    };

    scorecard.total += 1;

    if let Ok(talled_mon: Vec<u8>) = serde_json::to_vec(&scorecard) {
        redis::set(&address, key: &id.to_string(), value: &talled_mon) Result<(), Error>
            .map_err(op: |_| anyhow::anyhow!("Error saving to Redis"))?;
    }

    Ok(())
}
```

```
morsel_event::tally::Tally
pub ulid: String
```

Go to String

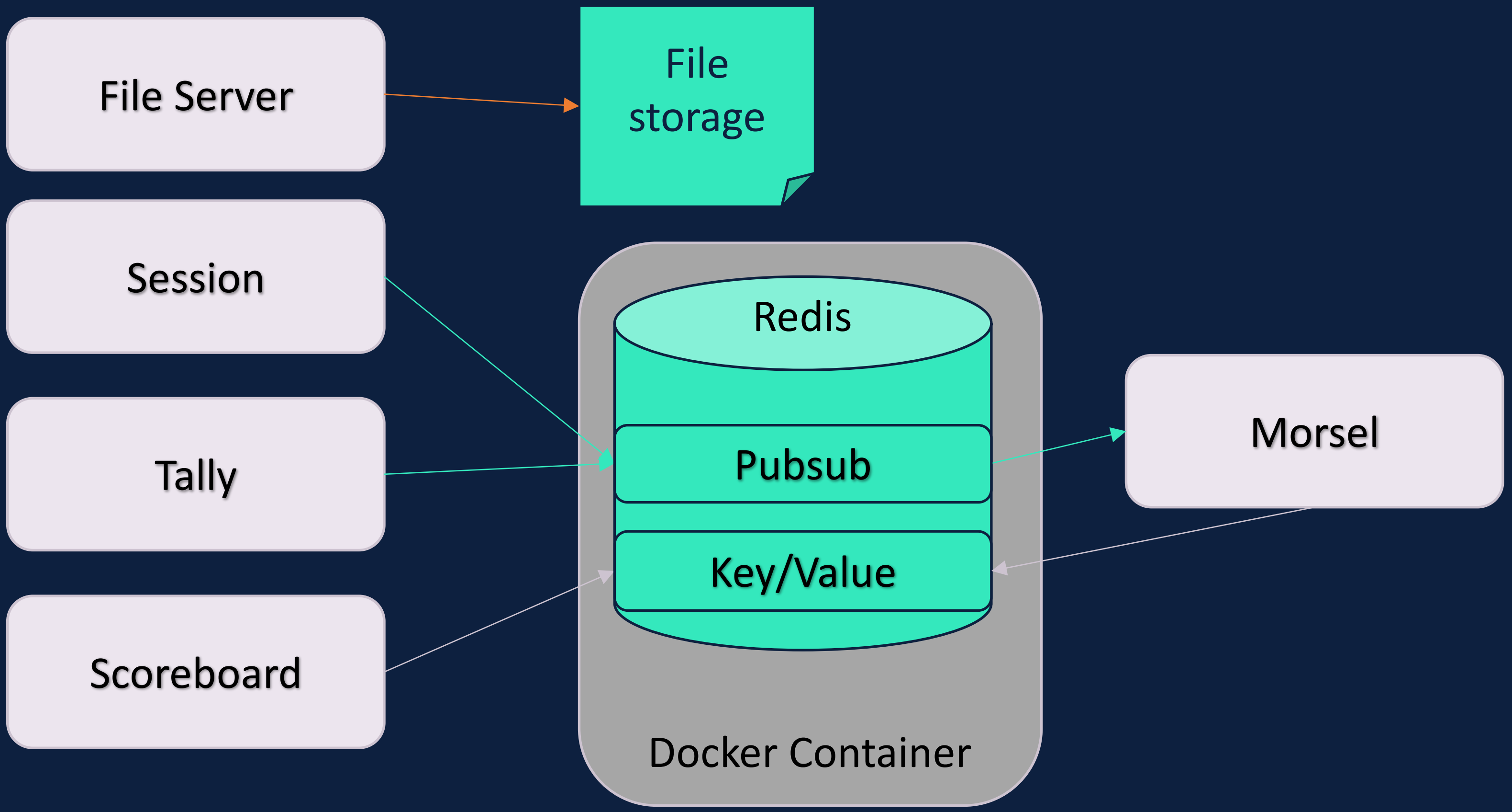
"ulid": Unknown word. cSpell

https://finickywhiskers.com is a small game designed to give a glimpse into our vision of the future.

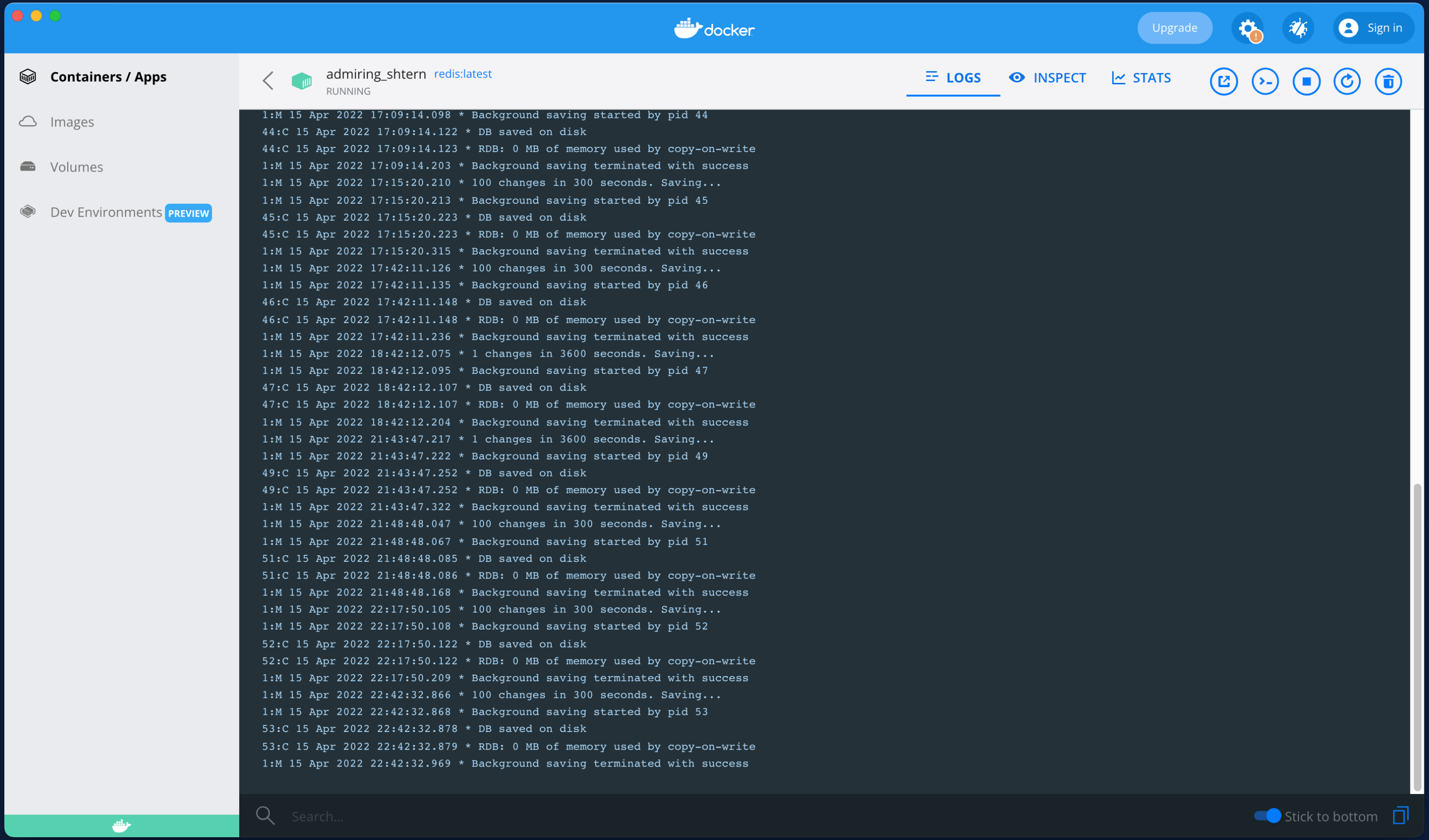
The game is composed of:

- One static front-end
- Six WebAssembly microservices
- Redis running in as a containerized service

Architecture



Behind the Game



```
1:M 15 Apr 2022 17:09:14.098 * Background saving started by pid 44
44:C 15 Apr 2022 17:09:14.122 * DB saved on disk
44:C 15 Apr 2022 17:09:14.123 * RDB: 0 MB of memory used by copy-on-write
1:M 15 Apr 2022 17:09:14.203 * Background saving terminated with success
1:M 15 Apr 2022 17:15:20.210 * 100 changes in 300 seconds. Saving...
1:M 15 Apr 2022 17:15:20.213 * Background saving started by pid 45
45:C 15 Apr 2022 17:15:20.223 * DB saved on disk
45:C 15 Apr 2022 17:15:20.223 * RDB: 0 MB of memory used by copy-on-write
1:M 15 Apr 2022 17:15:20.315 * Background saving terminated with success
1:M 15 Apr 2022 17:42:11.126 * 100 changes in 300 seconds. Saving...
1:M 15 Apr 2022 17:42:11.135 * Background saving started by pid 46
46:C 15 Apr 2022 17:42:11.148 * DB saved on disk
46:C 15 Apr 2022 17:42:11.148 * RDB: 0 MB of memory used by copy-on-write
1:M 15 Apr 2022 17:42:11.236 * Background saving terminated with success
1:M 15 Apr 2022 18:42:12.075 * 1 changes in 3600 seconds. Saving...
1:M 15 Apr 2022 18:42:12.095 * Background saving started by pid 47
47:C 15 Apr 2022 18:42:12.107 * DB saved on disk
47:C 15 Apr 2022 18:42:12.107 * RDB: 0 MB of memory used by copy-on-write
1:M 15 Apr 2022 18:42:12.204 * Background saving terminated with success
1:M 15 Apr 2022 21:43:47.217 * 1 changes in 3600 seconds. Saving...
1:M 15 Apr 2022 21:43:47.222 * Background saving started by pid 49
49:C 15 Apr 2022 21:43:47.252 * DB saved on disk
49:C 15 Apr 2022 21:43:47.252 * RDB: 0 MB of memory used by copy-on-write
1:M 15 Apr 2022 21:43:47.322 * Background saving terminated with success
1:M 15 Apr 2022 21:48:48.047 * 100 changes in 300 seconds. Saving...
1:M 15 Apr 2022 21:48:48.067 * Background saving started by pid 51
51:C 15 Apr 2022 21:48:48.085 * DB saved on disk
51:C 15 Apr 2022 21:48:48.086 * RDB: 0 MB of memory used by copy-on-write
1:M 15 Apr 2022 21:48:48.168 * Background saving terminated with success
1:M 15 Apr 2022 22:17:50.105 * 100 changes in 300 seconds. Saving...
1:M 15 Apr 2022 22:17:50.108 * Background saving started by pid 52
52:C 15 Apr 2022 22:17:50.122 * DB saved on disk
52:C 15 Apr 2022 22:17:50.122 * RDB: 0 MB of memory used by copy-on-write
1:M 15 Apr 2022 22:17:50.209 * Background saving terminated with success
1:M 15 Apr 2022 22:42:32.866 * 100 changes in 300 seconds. Saving...
1:M 15 Apr 2022 22:42:32.868 * Background saving started by pid 53
53:C 15 Apr 2022 22:42:32.878 * DB saved on disk
53:C 15 Apr 2022 22:42:32.879 * RDB: 0 MB of memory used by copy-on-write
1:M 15 Apr 2022 22:42:32.969 * Background saving terminated with success
```

As we click the buttons to feed the cat, we're executing request to Spin, which is spinning up a new WebAssembly instance to handle each request.

The faster you click, the more Wasm modules you start.

Finicky Whiskers is the world's most adorable manual load generator.

A big THANK YOU to the entire WebAssembly community!

- None of the work we have seen today would be possible without all the amazing people working on standardizing and implementing WebAssembly.

FERMYON

Game Over