A New Vue:
The Server Side WebAssembly/WASI Platform

Victor Adossi
Cosmonic
🗼 Tokyo

Vue Fes Japan 2025

📜 A brief, incomplete history of WebAssembly

ASM.js

C/C++ ➡️ JS (subset)

Emscripten

C/C++/… ➡️ LLVM ➡️ JS Subset/WebAssembly (“Core”) Module

Component Model *(“Modern” WebAssembly)*

C/C++/Rust/JS/… ➡️ LLVM/… ➡️ **WebAssembly Components**

🤷‍♂️ Why WebAssembly?

Security

No access to the outside world

Control over execution with fuel & epochs

Robustness

Multiple languages compile to Wasm

Wasm runtimes can run on multiple platforms(*)

Efficiency

Close to native execution

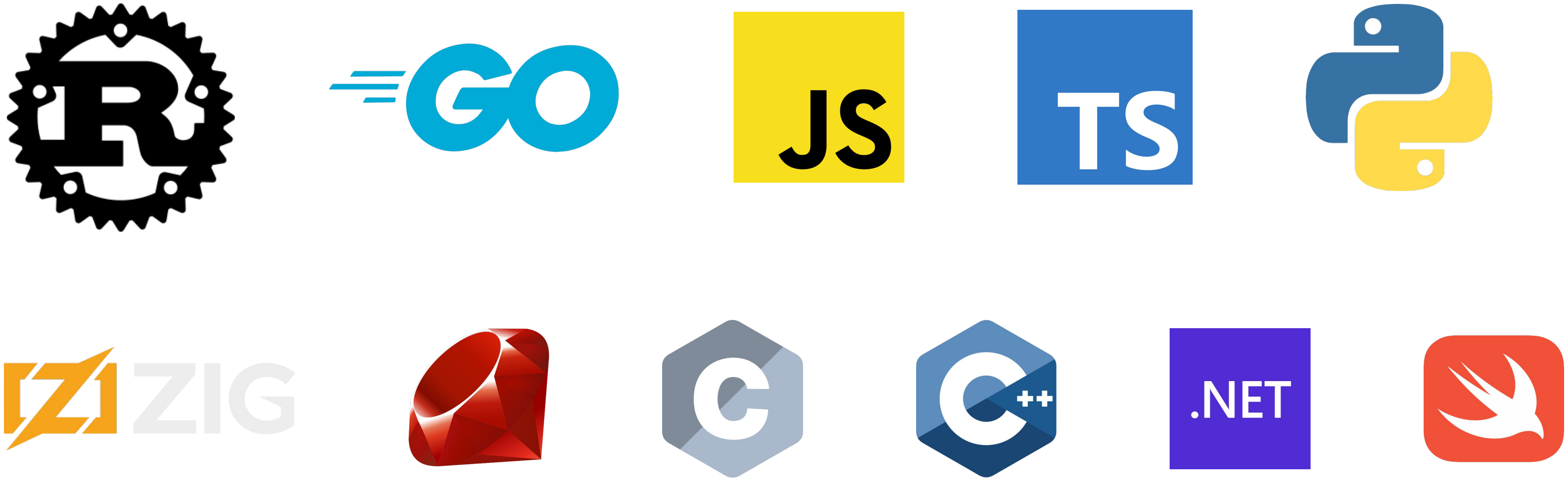
Small Binaries

Open Source, Open Standards

Work is done in the open

Stewarded by the Bytecode Alliance

🚀 Language Support is growing



Support varies by language & toolchain

LLVM languages usually get WASM support for ~free

🤷‍♂️ Why “Modern” WebAssembly?

Standards for interoperability and composition

How would you define a WebAssembly import for writing to `console.log`?

What if one WebAssembly module could call another one?

Asynchronously?

If everyone does this separately, is it possible to interoperate?

💡 **The Component Model enables standardized interoperability and composition for WebAssembly**

🤷‍♂️ Why “Modern” WebAssembly?

Rich Types via WebAssembly Interface Types (WIT)

If WebAssembly only knows `i32`, `i64`, `f32`, `f64` (*) how do we use `string` ?

```
package local:example;

interface greeter {
  greet: func(name: string) -> string;
}

world component {
  export greeter;
}
```

💡 **WebAssembly Interface Types (WIT) extend the Component Model with strong typing**

🤷‍♂️ Why “Modern” WebAssembly?

Components show you their capabilities

Need to [log | access ENV | read files | …] ? Ask and the nearby platform delivers.

```
package local:example;

interface greeter { ... }

world component {
  import wasi:cli/stdout;
  import wasi:cli/environment;
  export greeter;
}
```

💡 **By default, WebAssembly has access to none of the underlying platform, not even filesystem access.**

🦀 A short walk on the beach

A quick look at how this works in practice, in Rust

`component.rs`

```rust
mod bindings;

use bindings::Guest;

struct Component;

impl Guest for Component {
    fn greet(name: String) -> String {
        format!("Hello, {name}!")
    }
}

bindings::export!(Component)
```

`component.wasm`

```
(component
  (type (;0;)
      (instance
      (type (;0;) (tuple string string))
      (type (;1;) (list 0))
      (type (;2;) (func (result 1)))
      (export (;0;) "get-environment" (func
(type 2)))
      )
  )
  (import "wasi:cli/environment@0.2.0" …
   …
)
```

`cargo build --target=wasm32-wasip2`

Oh Right, we're at a Javascript conference

A quick look at how this works in practice, in ~~Rust~~ Javascript

component.js

```
export const greeter = {
    greet(name) {
        return `hello, ${name}`;
    }
};
```

component.wasm

```
(component
  (type (;0;)
      (instance
      (type (;0;) (tuple string string))
      (type (;1;) (list 0))
      (type (;2;) (func (result 1)))
      (export (;0;) "get-environment" (func
(type 2)))
      )
  )
  (import "wasi:cli/environment@0.2.0" …
   …
)
```

```
componentize-js -o component.wasm code.js
(jco componentize -o component.wasm code.js)
```

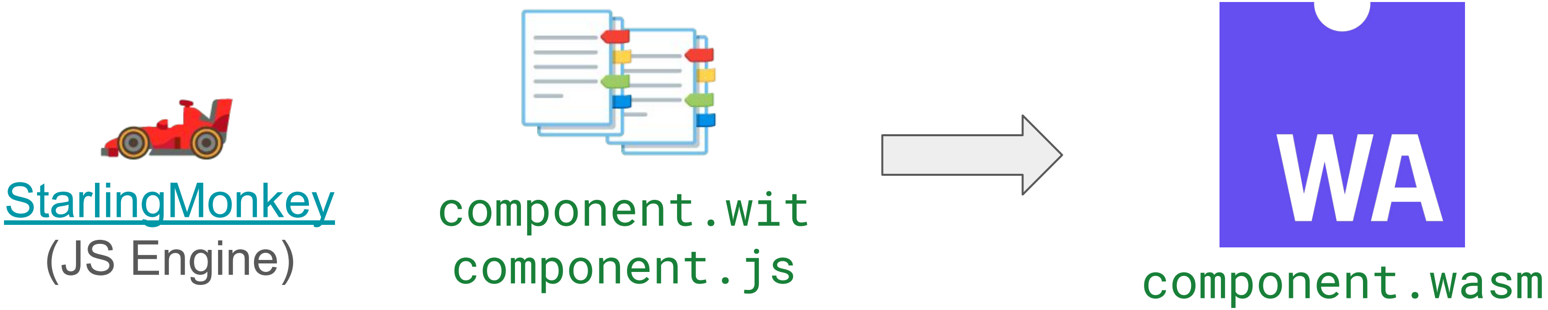
Oh Right, we’re at a **Vue** conference

Q: How does any of this relate to Vue?

A: Vue apps and components can be packaged as WebAssembly components, and run wherever WebAssembly runs

🤔 OK, How?

1. **Vue** is written in Javascript
2. SpiderMonkey is a Javascript Engine
3. **StarlingMonkey** is a fork of SpiderMonkey that compiles to WebAssembly
4. **We jam these two things together and make your code WebAssembly**



StarlingMonkey
(JS Engine)

```
component.wit
component.js
```

```
component.wasm
```

🤔 OK, but actually, how?

There are (at least) 3 ways to build your Vue apps & Components:

⭐ **WebAssembly HTTP Server for Client Side Rendered Vue code**

⭐⭐ **WebAssembly HTTP Server for Server Side Rendered Vue Code**

⭐⭐⭐ **Build your Vue (Vapor) component into a WebAssembly Component**

Scan this QR or go here 👇
github.com/t3hmrman/2025-vue-fes

⁉️ Wait, WebAssembly can make HTTP servers?

Yep – and it's ✨ standardized ✨

With the WebAssembly System Interface (WASI), you can write components that are web servers for any platform that supports the idea of serving HTTP.

StarlingMonkey also has support for `fetch-event`

```
addEventListener("fetch", (event) =>
  event.respondWith(
    (async () => {
      return new Response("Hello Web Standards!");
    })(),
  ),
);
```

Get the code

⭐ WebAssembly HTTP Server for Client Side Rendered Vue code

This is simple - all we have to do is serve a client bundle:

```
import { Hono } from "hono";
import { fire } from "hono/service-worker";

// Our entire client-side app has been reduced to
// this one entry point via vite-plugin-singlefile
import staticHTML from "../dist/app/index.html";

export const app = new Hono();
app.get("/", async (c) => c.html(staticHTML));

// Run the actual app, via fetch-event support
fire(app);
```

(TEMPT THE DEMO GODS)

Get the code

⭐⭐ WebAssembly HTTP Server for Server Side Rendered Vue Code

```
import { createSSRApp } from "vue";
import { renderToString } from "vue/server-renderer";

import clientJS from "../dist/app/ssr.js";
import clientCSS from "../dist/app/ssr.css";
import indexLayout from "./app/index.layout.html";

const vueApp = createSSRApp(App);

export const app = new Hono();
app.get("/app.js", async (c) => {
    c.header('Content-Type', 'application/javascript');
    return c.body(clientJS);
});

app.get("/app.css", async (c) => {
    c.header('Content-Type', 'text/css');
    return c.body(clientCSS);
});

app.get("/", async (c) => {
    const renderedApp = await renderToString(vueApp);
    const renderedHTML = indexLayout.replace("__APP_HTML__",
renderedApp);
    return c.html(renderedHTML);
});

fire(app);
```

```
// client code for hydration
import { createSSRApp } from
"vue";
import App from "./app/App.vue";
const app = createSSRApp(App);
app.mount("#app");
```

(PROVOKE THE DEMO GODS)

Get the code

⭐⭐⭐ Build your Vue (Vapor) component into a WebAssembly Component

Here’s the plan:

1. Shim the DOM related dependencies of Vue
2. Build Vue components into WebAssembly
3. Ship WebAssembly
4. ???
5. **Run everywhere WebAssembly runs**

(TRIUMPH OVER THE DEMO GODS)

Get the code

🕵️ What’s the catch?

🔌 Support for arbitrary `node:*` bultins hasn’t landed yet (Web standards ✅)
(We’re working on this, in more ways than one)

🚧 Our Browser WASI shim is still experimental
(It works, but the ecosystem has some better options)

💻 There is no native WebAssembly DOM support in browsers (yet?), but WebIDL can be converted to WIT and used

📦 JS WebAssembly binaries are somewhat large (for WebAssembly) – StarlingMonkey alone is currently around 10MB(*), Rust is xxxKB

🌏 The JS WebAssembly Ecosystem

Awesome people make the WebAssembly ecosystem work.

There are too many people to thank to put on just one slide, so I won't try.

There's lots more building to do – come join us!

BYTECODE ALLIANCE

@bytecodealliance/componentize-js

@bytecodealliance/jco

@bytecodealliance/StarlingMonkey

@bytecodealliance/wasmtime

🙇‍♂️終わり

Try out the examples!