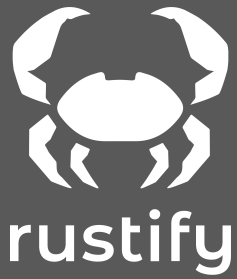
WebAssembly and Rust



Michael Allwright
michael.allwright@rustify.be

How many people here have used Docker?

“If WASM+WASI existed in 2008, we wouldn't have needed to create Docker. That's how important it is”

Solomon Hykes - 2019
Co-founder of Docker

Chapter 1

The origins of WebAssembly

Precursors to WebAssembly

Delivering rich applications over the web has been a pursuit since the 90s

- Required user to install a plugin
- Vendor lock-in/no portability
- Plagued with security issues

Microsoft® Silverlight™

Javascript

Only Javascript has stood the test of time

- Based on the ECMAScript standard
- Support from all major browsers
- Reasonable performance
- Awkward language with unusual semantics

WebAssembly (WASM)

A new **instruction set** for stack-based VMs

- Standardized by the W3C
- Supported in all major browsers
- Near native performance
- Secure & Portable
- Can be written in any language*

WebAssembly on the front-end

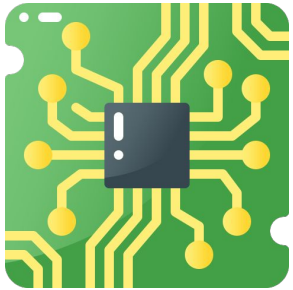
Web Applications

Libraries

WebAssembly on the back-end

WebAssembly enables serverless and Function as a Service (FaaS) on the backend

WebAssembly on the edge



WebAssembly can also be used on IoT devices

- Separate HAL from business logic
- Business logic is portable
- Light-weight containerization

Chapter 2

The WebAssembly System Interface

WebAssembly System Interface (WASI)

WebAssembly is a completely sandboxed technology

- All functionality must be imported or exported
- WASI defines standard interfaces for those functions

WebAssembly System Interface (WASI)

WASI is currently in its second preview

- Provides: IO, clocks, RNG, file systems, sockets, HTTP
- The third preview will introduce async support

Chapter 3

Using WebAssembly today

Compiling to WebAssembly

LLVM is one of two toolchains capable of generating WebAssembly*

- Used by both clang and rustc compilers
- Can only import/export functions with integers, floats, and booleans
- No support for strings, structs, threads

* The other toolchain is Binaryen and is used by the AssemblyScript language

Using WebAssembly from C/C++

The primary way to use WebAssembly from C/C++ is via **Emscripten**

Using WebAssembly from Rust

The primary way to use WebAssembly from Rust is via **wasm-bindgen**, which is **both** a crate and a tool for generating bindings

wasm-bindgen (crate)

The wasm-bindgen **crate**:

- Provides macros for creating bindings
- Is the foundation of the web-sys and js-sys crates
- Enables moving structs, strings etc. across the Rust/Javascript boundary

wasm-bindgen (tool)

After compiling to WebAssembly, the wasm-bindgen **tool** will:

- Strip the bindings information out of the compiled module
- Generate a Javascript loader

Chapter 4

When to use WebAssembly?

When to use WebAssembly?

On the front-end when:

- Full-stack web applications in Rust
- In browser transcoding, encryption, image processing, inference
- Porting existing applications to the web

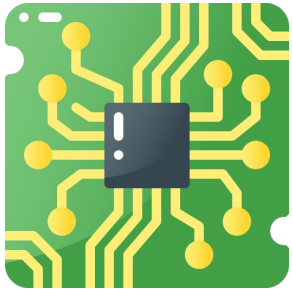
When to use WebAssembly?

On the back-end when:

- Full-stack web applications in Rust
- Running C/C++/Rust on FaaS infrastructure
- Computational loads

When to use WebAssembly?

On IoT devices to:



- Write portable business logic in a high level language
- Separate the HAL and business logic
- Light-weight containerization

Conclusion

Conclusions

- WebAssembly is actively used today
- Supported by all major browsers, on the backend, and on IoT devices
- Unlocks high performance, security, and portability

For training, consulting, or development in WebAssembly and/or Rust, feel free to contact me at: michael.allwright@rustify.be

Questions?