Rubyをこじらせて

Ruby is like a teenage angst to me

Uchio Kondo (@udzura)

- Affiliation: Mirrativ Inc.
- Hacker Supporter @Fukuoka City Engineer Cafe
- Advisor @Fjord Bootcamp
- Co-translator of "Leraning eBPF" (O'Reilly Japan)
- RubyKaigi Speaker (2016 ~)
- RubyKaigi 2019 Local Organizer @ Fukuoka

Recital of daily discipline

Prospectus

「目論見書」

Reflection of daily life

Bibliographic commentary

「解題」

2016

Haconiwa

Linux Container

Haconiwa is a Linux container

- Combine Linux container components from scratch
  - cgroup
  - namespace
  - pivot_root
  - capability, seccomp, ...

Linuxの要素技術を自分で繋ぎ直したのがHaconiwa

(2017..2018)

2019

Local Organizer @ Fukuoka

CRIU

CRIU

- Checkpoint and restore of processes (or containers)
  - Dump Rails' process status into files
  - Boot from it -> it's fast!

CRIUはプロセスをファイルにダンプして、そこから再生できるツール

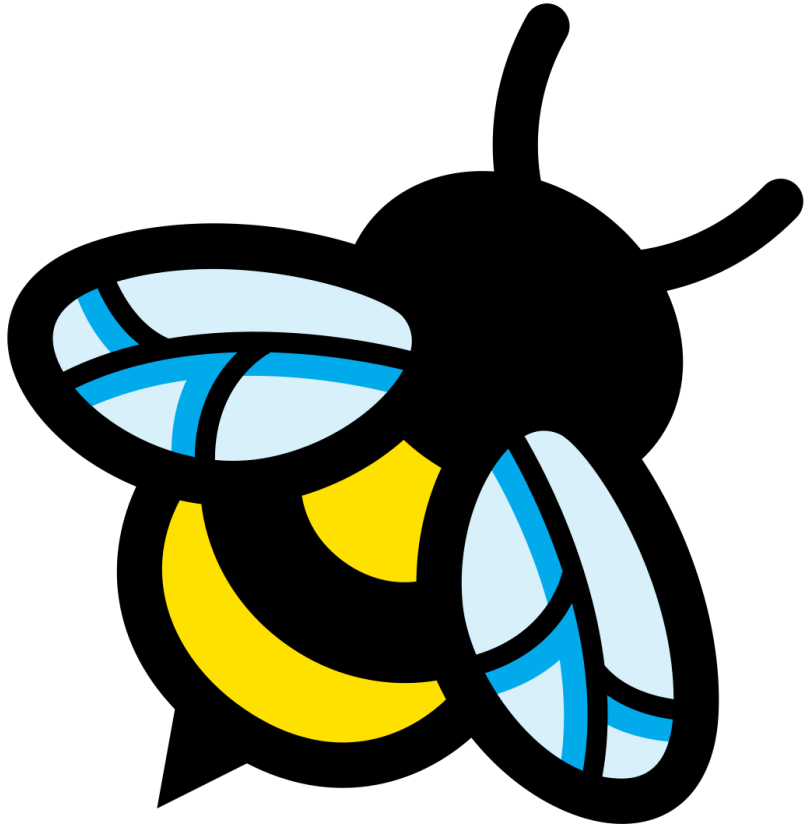
2021

eBPF

Rucy

eBPF

- Running programs with special format inside the Linux kernel
  - For Networking, Observability, Security...
  - Safer than kernel module
  - Deeper than system calls



eBPFは、カーネルの機能を使うための仕組みの一つ
カーネルモジュールより安全、システムコールより奥深い

What is Rucy

- Rucy **compiles Ruby scripts** into special bytecodes
  - The eBPF bytecodes!
  - Rucy = Ruby Compiler = RuC

RucyはRubyスクリプトをeBPFの形式に「コンパイル」する

2022

eBPF (RbBCC)

How is Rucy different from RbBCC?

Name	Strategy	Detail
RbBCC	JIT	Is an FFI to libbcc / Rusy Assoc. Grant
Rucy	AOT	Compiles Ruby scripts into eBPF via mruby bytecodes

How are they different? (details)

- Basically, eBPF is moving towards a AOT-compiled ecosystem (BPF CO-RE)
  - So the future is Rucy's side
- RbBCC has a larger coverage of eBPF functions
  - Sufficient enough for learning and small tools

今の潮流はAOTだが、サクッと使う分にはRbBCCの方が機能が多い

[PR]

- This book will help your understanding
- Thanks to Tori-san, it is pleasant to read!

鳥井さん、ありがとうございました

~~2023~~

敗北を知りたい

2024

WebAssembly

WebAssembly

- As you know, a technology for running code in browsers
  - Only browsers?

「なんかブラウザで動くやつ」

WebAssembly is not only for browsers

- Browsers are just "one of the runnable environments"
- Can run everywhere with portable VM
  - envoy, fluent-bit, Containers like youki...
  - even for real embedded systems...

ブラウザ外の「アプリケーション組み込み」が個人的に熱い。
envoy、youki、使ったことありますか？

The "Web" part of WebAssembly

- WebAssembly will probably not be just for the web
- ... Just like eBPF is no longer "Berkeley Packer Filter"

Why Ruby for WebAssembly (again)

- Ruby for WebAssembly, with more "embeddability"
  - Also with the mruby.

違うアプローチをしたい理由があります

Talk about this again later

Trends of thoughts

Trends

- So-called low-level technologies?
- Linux mania? (it's coincidentally)
- "I tried utilizing `${mysterious_tech}` from Ruby!"

「`${謎技術}` をRubyから使ってみた！」

My fighting style

「芸風」

Give a jab to the lower layers from Ruby World

低めのレイヤーにRubyでいっちょ噛み

Are you interested in low layers?

- It's even "unknown unknown" from ordinary web application engineers...

普通にWebアプリを作ってる分には隠蔽されていることばかり...

Why low-level?

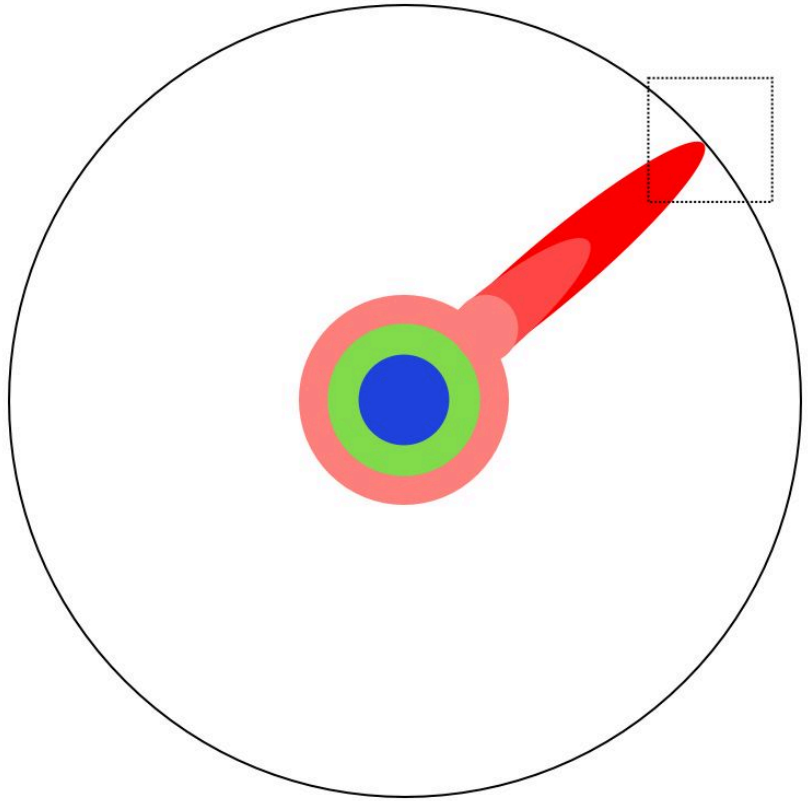
A typical low-layer project...

- Has NO Ruby bindings
- Has NO Ruby sample code
- Just has samples for like Go, Python, C++, and Rust

最近の低レイヤツール、クラウドネイティブミドルウェア、マジでRubyのサポートがない（個人の感想です）

low-layer is SOTA (state-of-the-art)

- Desire to touch SOTA
- I want to use Ruby even when touching advanced things
  - (image: Link)



Rubyから触れない＝ `${最新の超技術X}` がアウトオブ眼中になる、はなんか勿体無い

How am I satisfied with "Getting Started"?

- Understand the essence through Ruby
  - Adapting SOTAs to Ruby == Hack

Rubyサポートを入れるには「本質」が分かってないといけないので勉強になる

Hack the SOTA technologies

Create something useful || interesting

Back to my newest idea

mruby/edge in Depth

WebAssembly in Depth

Core WebAssembly

- Key specifications:
  - import/export Functions
  - Linear memory
- I'll omit the latter for today...

Linear memoryは線形メモリとも / その話は今日は時間なし...

Making a smallest WebAssembly binary

- Written in C

```
// sample.c
#include <emscripten.h>

void log_something(int size);

int EMSCRIPTEN_KEEPALIVE calc_plus(int n, int m) {
  log_something(n + m);
  return 0;
}
```

Compile it with emscripten

```
$ emcc -o sample.wasm --js-library ./lib.js --no-entry ./sample.c
cache:INFO:  - ok
```

- ※ lib.js is here

```
mergeInto(LibraryManager.library, {
    log_something: function(value) { /* TODO */ }
});
```

Export section

```
$ wasm-objdump -x -j Export sample.wasm

sample.wasm:    file format wasm 0x1

Section Details:

Export[10]:
 - memory[0] -> "memory"
 - func[2] <calc_plus> -> "calc_plus"
 - table[0] -> "__indirect_function_table"
 - func[3] <_initialize> -> "_initialize" ...
```

- Invoke `calc_plus()` from "browser"

Import section

```
$ wasm-objdump -x -j Import sample.wasm

sample.wasm:    file format wasm 0x1

Section Details:

Import[1]:
 - func[0] sig=2 <env.log_something> <- env.log_something
```

- Inject `log_something()` "browser" function into wasm instance

How to use in browser

```
const obj = {
  env: {
    // Specify the browser-side function here
    log_something: function(value) {
      let log = "sample wasm! 12 + 34 = " + value.toString();
      document.getElementById("placeholder").innerText = log;
    }
  },
};

WebAssembly.instantiateStreaming(fetch("./sample.wasm"), obj).then(
  (obj) => {
    // Call the function defined in wasm here
    obj.instance.exports.calc_plus(12 + 34);
  },
);
```

Live demo

Output: {{here}}

The primary concept of WebAssembly

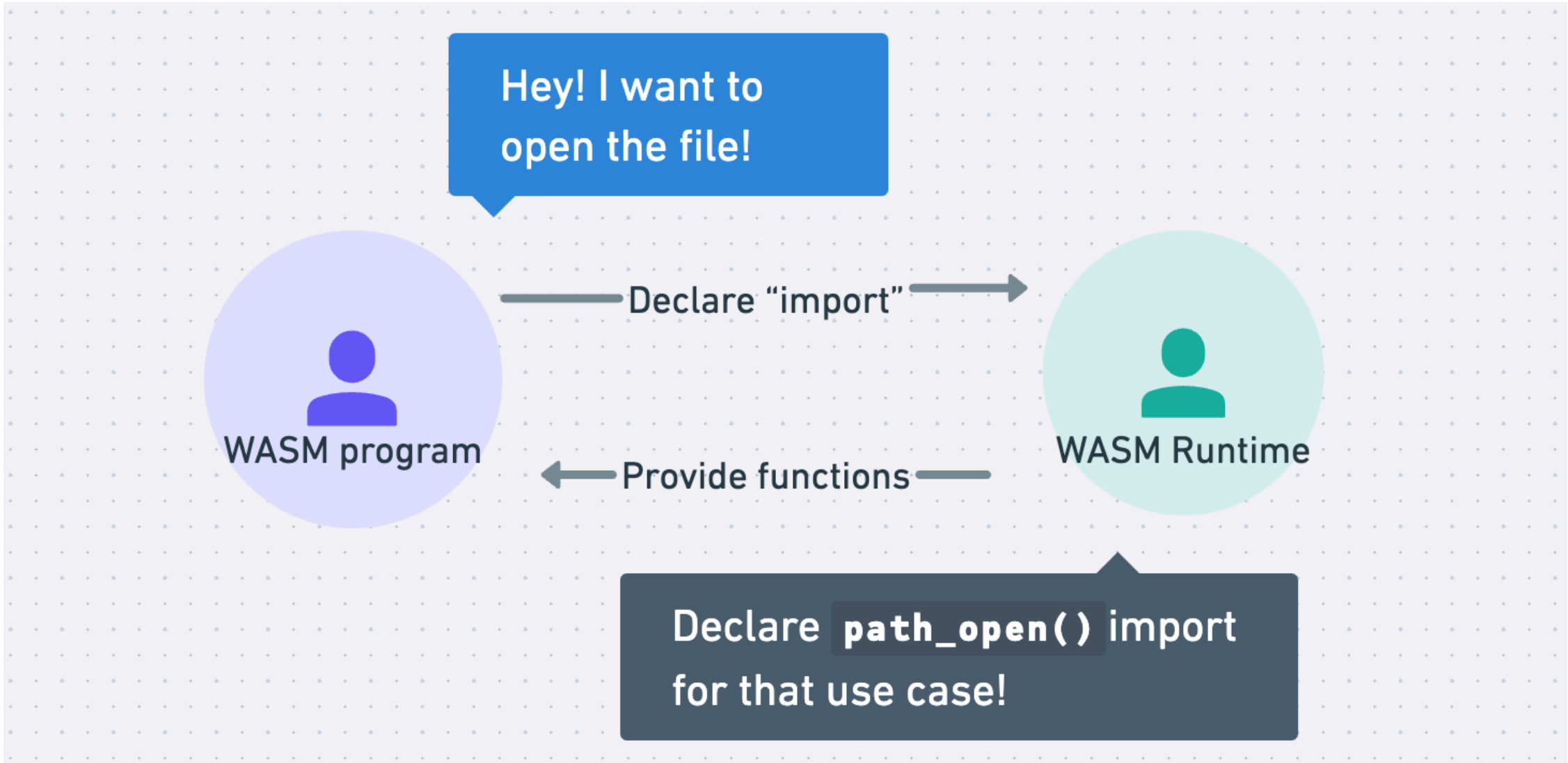
昭和の表現で言えば「一丁目一番地」の仕様やで（個人の意見です）

Simple

WASI is simple if you understand import/export

- WASI = A set of usable functions to `import`
  - For cooperation with the system in a nice way

WASI = 「これをimportして使えばシステム操作がええ感じにできるで」という関数のセット


Hey! I want to open the file!
Declare “import”
WASM program
WASM Runtime
Provide functions
Declare path_open() import for that use case!

e.g. Emulating WASI in a browser

- https://github.com/bjorn3/browser_wasi_shim
- Implement the "system call" of `random_get` in browser JS

```
random_get(buf: number, buf_len: number) {
    const buffer8 = new Uint8Array(...);
    // omit stuffs...
    for (let i = 0; i < buf_len; i += 65536) {
        crypto.getRandomValues(buffer8.subarray(i, i + 65536));
    }
}
```

`random_get` をJSで書いてimportさせればWASMでrandomが使えるということ

What I expect with mruby/edge

- To export Ruby method definitions as they are
- To use imported functions as Ruby-level methods

Write it straightforward

素直にメソッドを書いたらexport/importされて欲しい

Code image (to be implemented)

- This doesn't mean it will be implemented exactly this way...

```ruby
# @export!
# @rbs (Integer) -> Integeer
def fib(n)
  # ...
end

# @import!
# @rbs (String) -> void
def console_log(str) = __imported__!

def _start
  console_log("Hello World " + fib(10).to_s)
end
```

One More Thing

Future of WebAssembly

Component Model

WebAssembly Component Model

- Refer to interface of Core WASM
  - It's a bit fuzzy - like a C dynamic object
- Be more convenient to "connect" programs and world

Core WASM の仕様は色々余地があるので、型をしっかりし、自動検知や自動生成にフレンドリーにした感じという理解でOK、だと思う

Tools for the WebAssembly Component Model

- User-friendly typing system
  - WIT format
- Binary specification
  - Canonical ABI
- Toolchains
  - Once the above is generally available

WIT format

```
world rubyist-kaigi {
    export fib: func(n: i32) -> i32;

    import console_log: func(log: string);
}
```

- OK, some kinda DSL may be desired

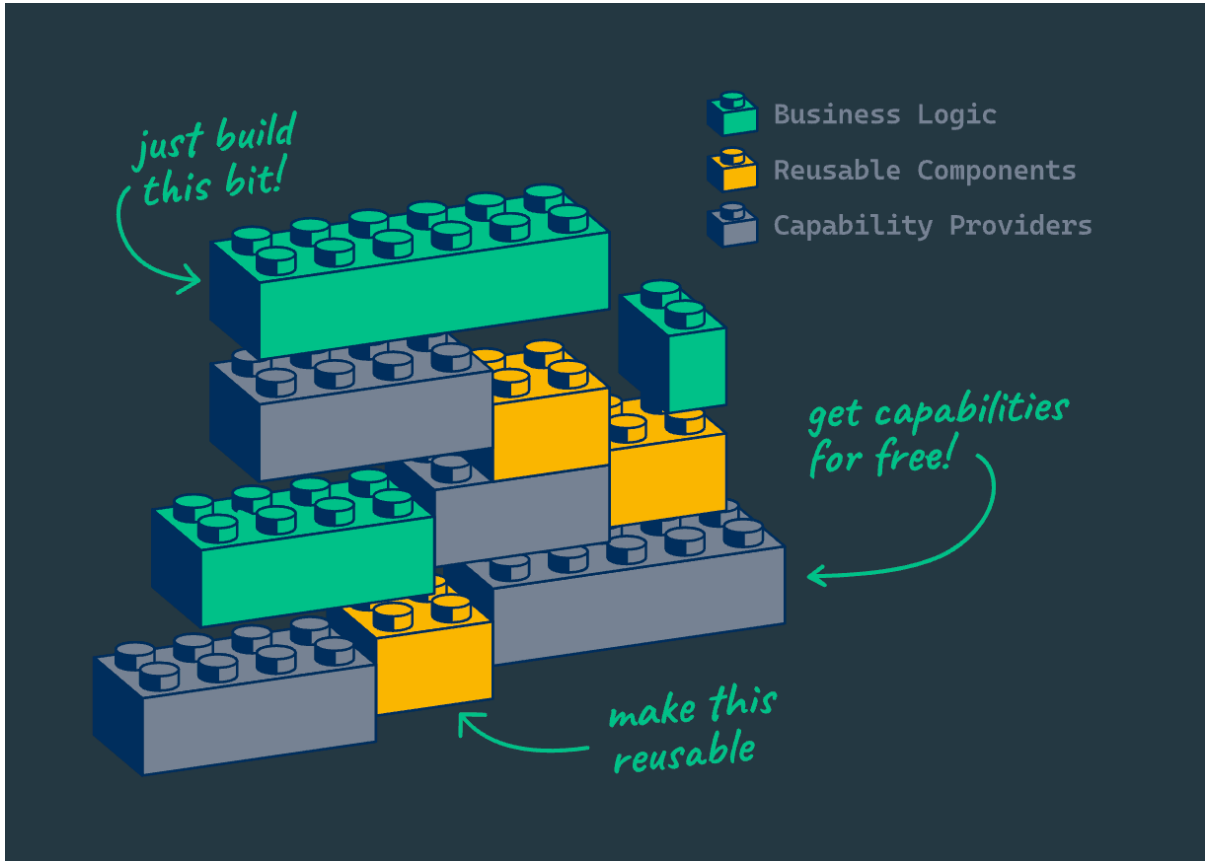
Rubyist的には...

FYI: Understanding by Hands-on

- "手書きで理解するWebAssembly Component Model"
  - (Understanding the WebAssembly Component Model by Hand Assemble)
  - https://nulab.com/ja/blog/nulab/webassembly-component-model/
  - https://nulab.com/ja/blog/nulab/webassembly-component-model-hello-world/

wasmCloud, Future for example

- https://wasmcloud.com/
- A CNCF Sandbox Project



Running Ruby on wasmCloud

> If you prefer working in a language that isn't listed here, let us know!

- https://wasmcloud.com/docs/tour/hello-world?lang=unlisted

Example of running mruby/edge (roughly)

```
use mrubyedge::{mrb_helper, vm::RObject};
// ...
impl Guest for HttpServer {
    fn handle(_request: IncomingRequest, response_out: ResponseOutparam) {
        let write_response = |body: &str| { ... };
        let bin = include_bytes!("./fib.mrb");
        let rite = mrubyedge::rite::load(bin).unwrap();
        let mut vm = mrubyedge::vm::VM::open(rite);
        vm.prelude().unwrap(); //...
        match mrb_helper::mrb_funcall(&mut vm, &top_self, "fib".to_string(), &args) {
            Ok(val) => { write_response(&val) }
            Err(ex) => { dbg!(ex); }
        } //...
    }
}
```

- https://github.com/udzura/mruby-wasmcloud-http

Creating a WASM binary that contains mruby

```
$ wash build
   Compiling http-hello-world v0.1.0 (/home/ubuntu/mrubyhttp)
    Finished `release` profile [optimized] target(s) in 0.29s

Component built and signed and can be found at "/../build/http_hello_world_s.wasm"

$ # The mruby binary is embedded
$ strings build/http_hello_world_s.wasm | grep MATZ
MATZ0000IREP

$ wasm-tools component wit build/http_hello_world_s.wasm | head -n 20
package root:component;

world root {
  import wasi:clocks/monotonic-clock@0.2.0;
  // ....

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

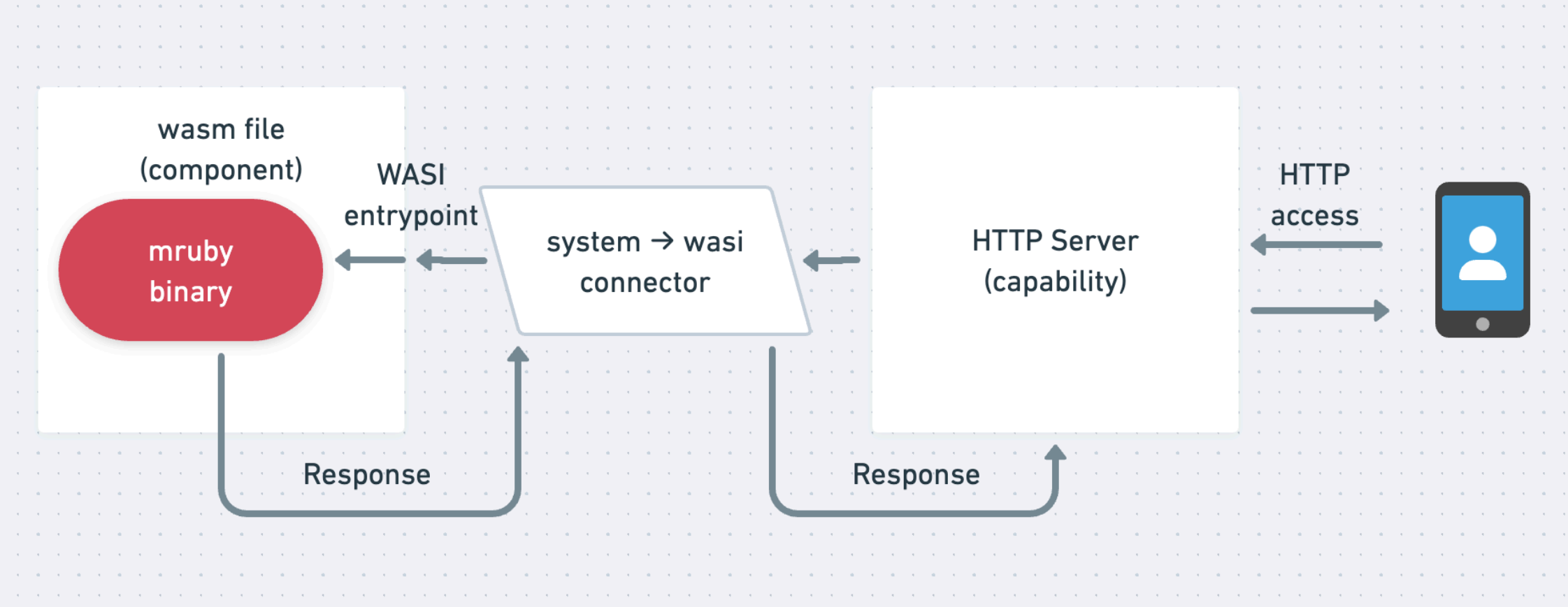
Running this WASM on wasmCloud

```
$ wash app deploy wadm.yaml
$ wash app status rust-hello-world

rust-hello-world@ - Deployed
  Name                                          Kind           Status
  http_component                                SpreadScaler   Deployed
  httpserver -(wasi:http)-> http_component      LinkScaler     Deployed
  httpserver                                    SpreadScaler   Deployed

$ curl localhost:8080
fib(15) = 610
```

wasmCloud concepts



Ruby wants to connect the world, too

Rubyでコアビジネスロジックを書く、特殊なアルゴリズムとかLLMなところとかクラウドネイティブな機能は他の言語のものと組み合わせる、という世界観。例えば

mizchi
@mizchi · フォローする

.@yu_suke1994 と tskaigi で話した Ruby がこの先生きのこるにはという話、早いとこ wasm 対応の軽量ランタイム出して warg.io かなんかで component model エコシステムと統合されるしかねえ、という話になったので、@udzura さん頑張ってほしい

warg.io
warg
A secure registry protocol for Wasm packages

午前11:48 · 2024年5月17日

40 返信 リンクをコピー

2件の返信を読む

> I talked with @yu_suke1994 at tskaigi about the future of Ruby, and we agreed that the only way for Ruby to survive is to release a lightweight runtime that supports wasm soon and integrate it with the component model ecosystem. So, @udzura, please do your best!

Wrapping up

See you in Matsuyama!