cosmonic

Distributed WebAssembly with wasmCloud, WIT and wRPC

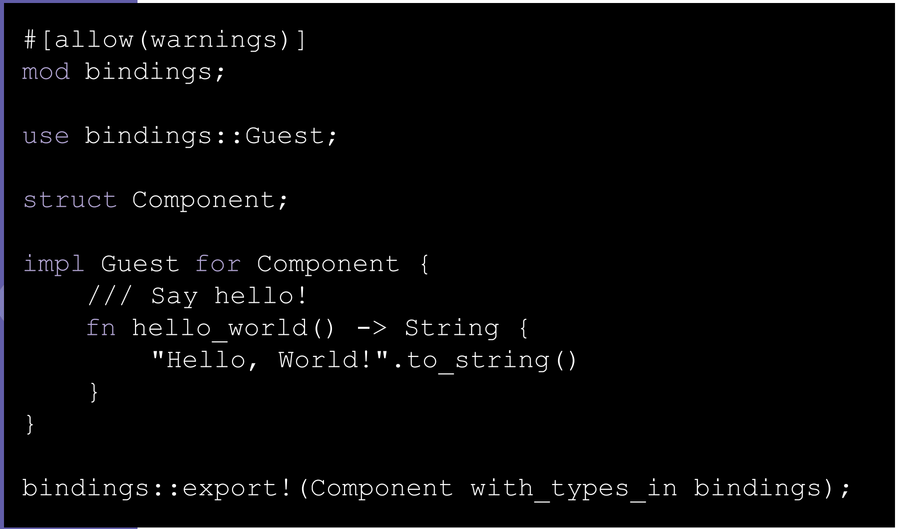
Victor Adossi
Cosmonic

A quick recap

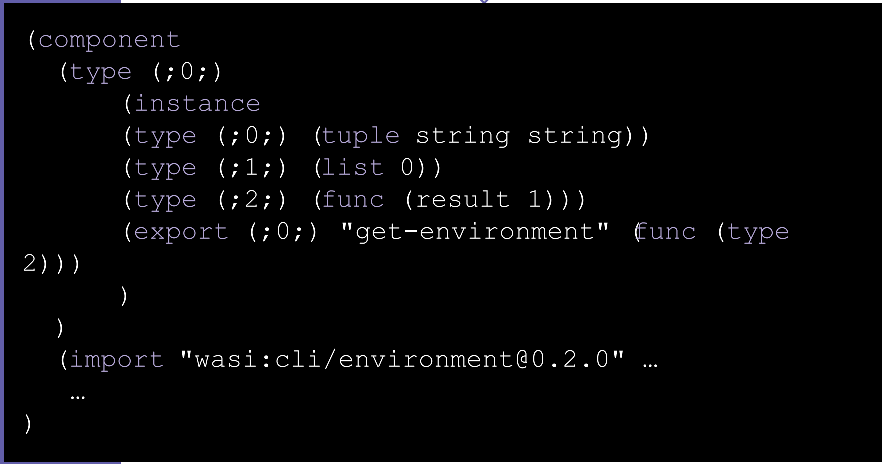
You write your $LANGUAGE code

Your code compiles to a WebAssembly binary

Your $LANGUAGE compiler builds in support for WebAssembly System Interface (WASI p1/p2/p3) so you write high level types, read files, use ENV variables etc.

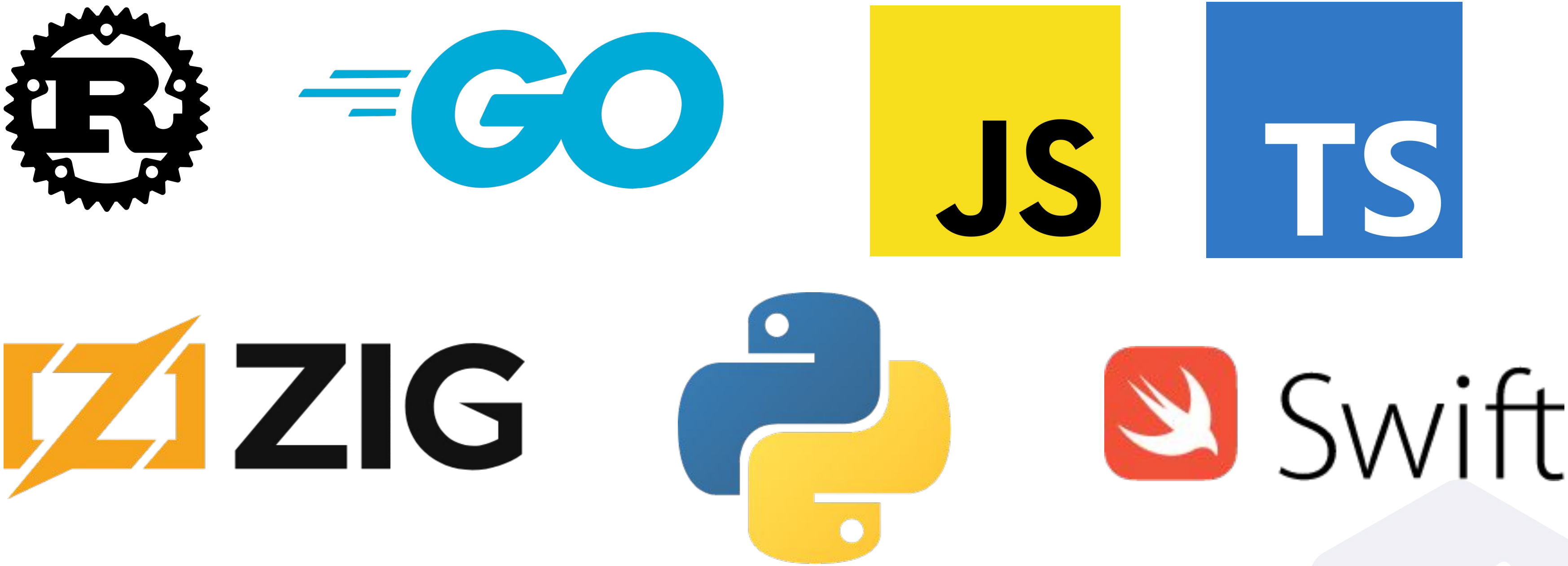


```
#[allow(warnings)]
mod bindings;

use bindings::Guest;

struct Component;

impl Guest for Component {
    /// Say hello!
    fn hello_world() -> String {
        "Hello, World!".to_string()
    }
}

bindings::export!(Component with_types_in bindings);
```



```
(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" …
   …
)
```

Language Support



Building on wasmtime for more functionality

`wasmtime` is **great** at running WebAssembly

It doesn't (and shouldn't) do everything, app and infra developers should not use it directly.

We must add features, ergonomics, scale, and more *on top* of `wasmtime` if we want more from WebAssembly

If `wasmtime` is like `java` – JRE what is the equivalent of Spring Boot?

Enter wasmCloud, a Wasm compute mesh

We built **CNCF wasmCloud**, a F/OSS (Apache v2) platform for building systems with WebAssembly

`wasmcloud` builds on `wasmtime` which means you can more easily build and test and run WebAssembly apps, solve ecosystem problems like logging, database access, and more in a consistent way.

wasmcloud

Does WebAssembly compose?

We have binaries that can do some computations and high level language functionality.

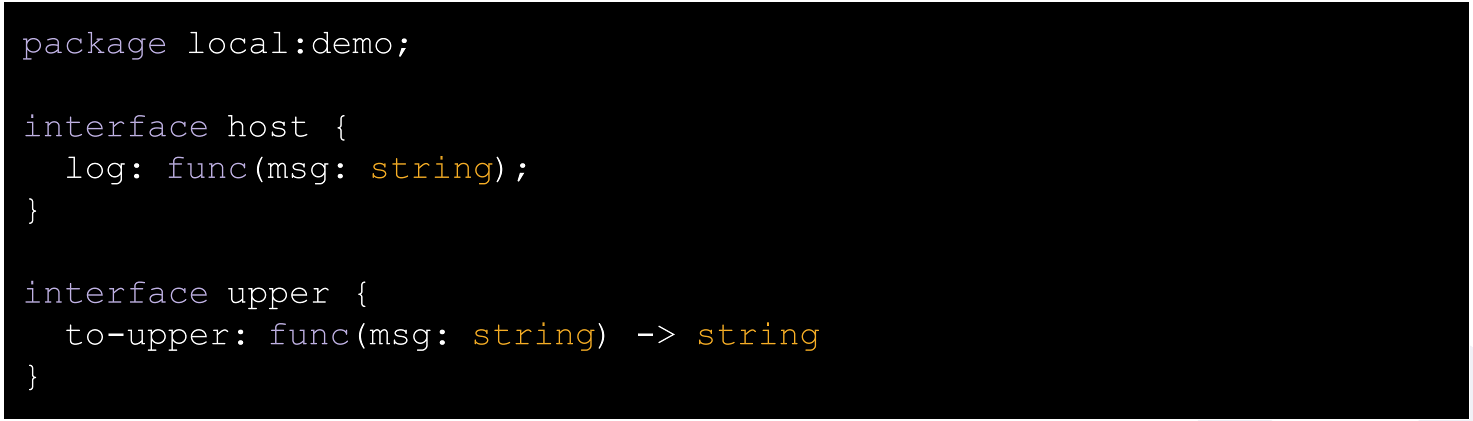
How do we combine two WebAssembly components that do small things to do larger tasks (abstraction)?

The obvious ecosystem answer is composing and creating new components (i.e. `wasm-tools compose`)

WebAssembly Interface types (WIT)

Before we can *compose* functionality, we need a way to *describe* functionality.

WIT is a interface definition language (IDL, ex. gRPC/OpenAPI/Thrift/etc) – that works at the binary level, no network required.



```
package local:demo;

interface host {
  log: func(msg: string);
}

interface upper {
  to-upper: func(msg: string) -> string
}
```

Does WebAssembly compose? (pt 2)

Composing two components into the same component works *most* of the time, but now they must be built together and deploy together.

Let’s borrow some concepts from:

- Every application on the internet?
- ESB
- Micro services
- APIs

What if we had a RPC system for WebAssembly?

wRPC: a Wasm-native RPC protocol

We developed and contributed wRPC, a Wasm-native RPC protocol to the Bytecode Alliance

You write WIT (just like 2 slides ago), and we take your binary and turn it into something callable over the network when running in wasmCloud

BYTECODE
ALLIANCE

What about programs that don’t run in Wasm yet?

Not all the critical functionality you need can run in the WebAssembly sandbox today

For those things, we have **wasmCloud Providers** we take a regular binary (ex. Rust, Golang), enable it to *respond* to WIT contracts

Example: our SQL Postgres Provider

Preparing WebAssembly to scale and more

We work on wasmCloud so that WebAssembly can scale up across computers, regions, clouds and more.

We work in the open and we'd love for you to join us and the Bytecode Alliance in building out the ecosystem!

Thanks for listening!