An experimental outbound HTTP library for the WebAssembly System Interface

March 8, 2021

This article originally appeared on the DeisLabs blog

Over the last year, our team has been experimenting with executing WebAssembly workloads on the server using the WebAssembly System Interface and Wasmtime. While more and more proposals get implemented, networking is one scenario that doesn't have a stable API yet. This limits current applications compiled to WASI, and restricts the types of workloads that can be executed in WASI runtimes.

Today we are releasing a library which adds support for outgoing HTTP requests for WASI modules running in Wasmtime. This is an experiment intended to provide a *temporary* workaround until the WASI networking API is stable, and is compatible with Wasmtime v0.24 by using the `wasi_experiemental_http_wasmtime` crate. We expect that once the WASI sockets proposal gets adopted and implemented in language toolchains, the need for this library will vanish.

Using the new HTTP library

There are two main components to this project - libraries to help building Wasm modules, and a library that adds runtime support for instantiating those modules in Wasmtime. Let's first see how to build a module that sends an HTTP request in Rust:

```
use bytes::Bytes;
use http;
use wasi_experimental_http;

#[no_mangle]
pub extern "C" fn _start() {
    let url = "https://<some-domain>/post".to_string();
    let req = http::request::Builder::new()
        .method(http::Method::POST)
        .uri(&url)
```

```
        .header("Content-Type", "text/plain")
    let b = Bytes::from("sending a body in a POST request");
    let req = req.body(Some(b)).unwrap();

    let res = wasi_experimental_http::request(req)
        .expect("cannot make request");
    let str = std::str::from_utf8(&res.body())
        .unwrap()
        .to_string();
    // do something with the body
}
```

Because we are reusing the Rust `Request` and `Response` structures from the `http` crate (with `Bytes` request and response bodies), this way of building an HTTP request should feel familiar to Rust developers. Then, the request can be sent using the `request` function from the `wasi_experimental_http` crate, which returns a response that can be read appropriately, depending on the content type of the response body. This simple program can now be compiled to the `wasm32-wasi` target.

AssemblyScript is another popular language that natively compiles to WebAssembly, and the library we are releasing also includes an AssemblyScript NPM package (with examples for how it can be used on GitHub).

To run the module in a WebAssembly runtime, we have to satisfy the new functionality, and we can do this in Wasmtime using the `wasi_experimental_http_wasmtime` crate:

```
use wasi_experimental_http_wasmtime;
use wasmtime::*;
use wasmtime_wasi::Wasi;

let store = Store::default();
let mut linker = Linker::new(&store);
let wasi = Wasi::new(&store, ctx);

// link the WASI core functions
wasi.add_to_linker(&mut linker)?;

// link the experimental HTTP support
wasi_experimental_http_wasmtime::link_http(&mut linker, None)?;
```

The Wasmtime implementation also enables passing a list of allowed hosts - an optional and configurable list of domains or hosts that guest modules are allowed to send requests to. If a guest module attempts to send a request to a domain not explicitly allowed when the runtime was configured, it receives an error:

```
'cannot make request:
  URL not allowed because domain or subdomain not in allowed list
```

Finally, this library works in a similar way to WASI, or to libraries built on top of WASI: it adds a new Wasm import module, `wasi_experimental_http`, with a single import function, `req`, which receives the buffer sources for the request data, performs the request on the host, then writes the response data back to the guest module's memory, which can then be used by the guest module.

Support for this library

We are in the process of adding support for this library and outgoing HTTP requests in Krustlet and WAGI, so keep an eye out in the respective pull request queues.

We are also happy to help those who want to add support for this library in their projects.

Known limitations and contributing

While this library can currently be used to send and receive HTTP data, there are currently some limitations (for an updated list, please check the issue queue in the repository):

- there is no support for streaming HTTP responses, which this means guest modules have to wait until the entire body has been written by the runtime before reading it.
- there are no WITX definitions, which means we have to manually keep the host call and guest implementations in sync. Adding WITX definitions could also simplify adding support for other WASI runtimes (the only WebAssembly runtime currently supported by this library is Wasmtime).
- this library does not aim to add support for running HTTP servers in WebAssembly.

We welcome all contributions that adhere to the Microsoft Open Source Code of Conduct. Feel free to create pull requests or open issues with any feature suggestion, and we are happy to chat about the project in the WebAssembly Discord server, or in the ByteCodeAlliance Zulip chat.