
KubeCon
CloudNativeCon
North America 2022
BUILDING FOR THE ROAD AHEAD
DETROIT 2022

WasmEdgeRuntime

KubeCon | CloudNativeCon
North America 2022

BUILDING FOR THE ROAD AHEAD

DETROIT 2022

Cloud-native WebAssembly: Containerization on the edge

Michael Yuan, Second State & WasmEdge

Cloud-native WebAssembly

KubeCon | CloudNativeCon
North America 2022
BUILDING FOR THE ROAD AHEAD
DETROIT 2022

KubeCon | CloudNativeCon
North America 2022
BUILDING FOR THE ROAD AHEAD
DETROIT 2022

October 24-28, 2021

WasmEdge brings cloud-native tooling to WebAssembly (e.g., Docker & K8s)

https://hackmd.io/@wasmedge/SJCmYPDmo

Create a database-driven HTTP microservice in WebAssembly

https://github.com/second-state/microservice-rust-mysql

Develop a Dapr-based service mesh in WebAssembly

https://github.com/second-state/dapr-wasm

How it started

Container-based service mesh

- Managed by k8s or similar
- Microservice in a pod
  - Sidecar container
  - Application container
- SDK or proxy injection
- Istio, Linkerd, Cilium, Dapr

Pain points

- Heavy weight
- Slow (esp startup)
- Unsafe
- NOT portable across platforms
- Proxy has a lot of CPU overhead

How it’s going

Opinionated runtime optimized for microservices

- 1/100 the size of typical LXC images
- 1000x faster startup time
- Near native runtime performance
- Secure by default and very small attack surface
- Completely portable across platforms
- Programming language agnostic
- Plays well with k8s, service mesh, distributed runtimes etc.

Trade-offs

Too good to be true? There is no free lunch.

- Not a general OS environment
- Must learn new language SDKs to create optimized services
- Common libraries need to be ported

WasmEdgeRuntime

A lightweight, secure, high-performance and extensible WebAssembly Runtime

1. Support networking socket and web services
2. Support databases, caches, and DOs
3. Support AI inference in Tensorflow, OpenVino, PyTorch etc.
4. Seamlessly integrates into the existing cloud-native infra
5. Support writing wasm programs using JS

https://github.com/WasmEdge/WasmEdge

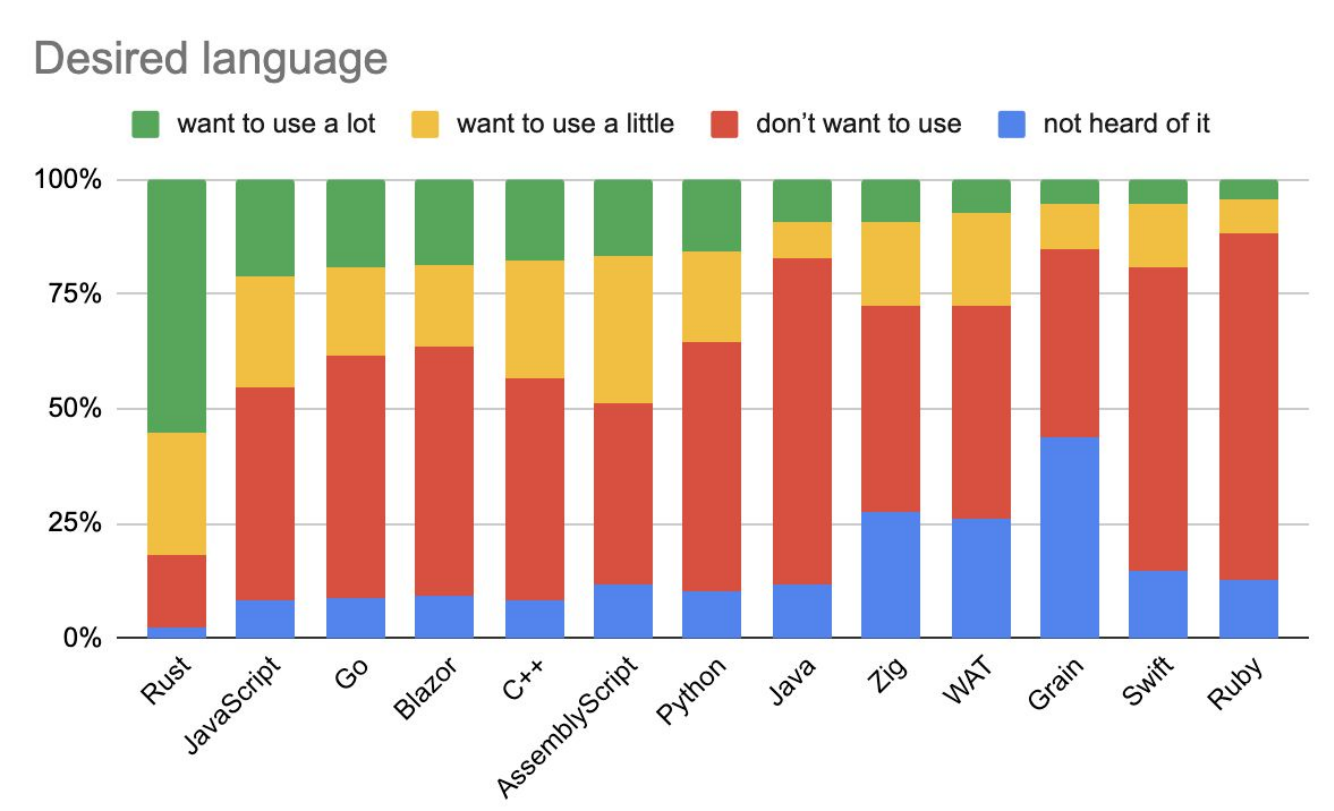
WasmEdgeRuntime

KubeCon | CloudNativeCon North America 2022

BUILDING FOR THE ROAD AHEAD

DETROIT 2022

Language support for Cloud-native WebAssembly apps



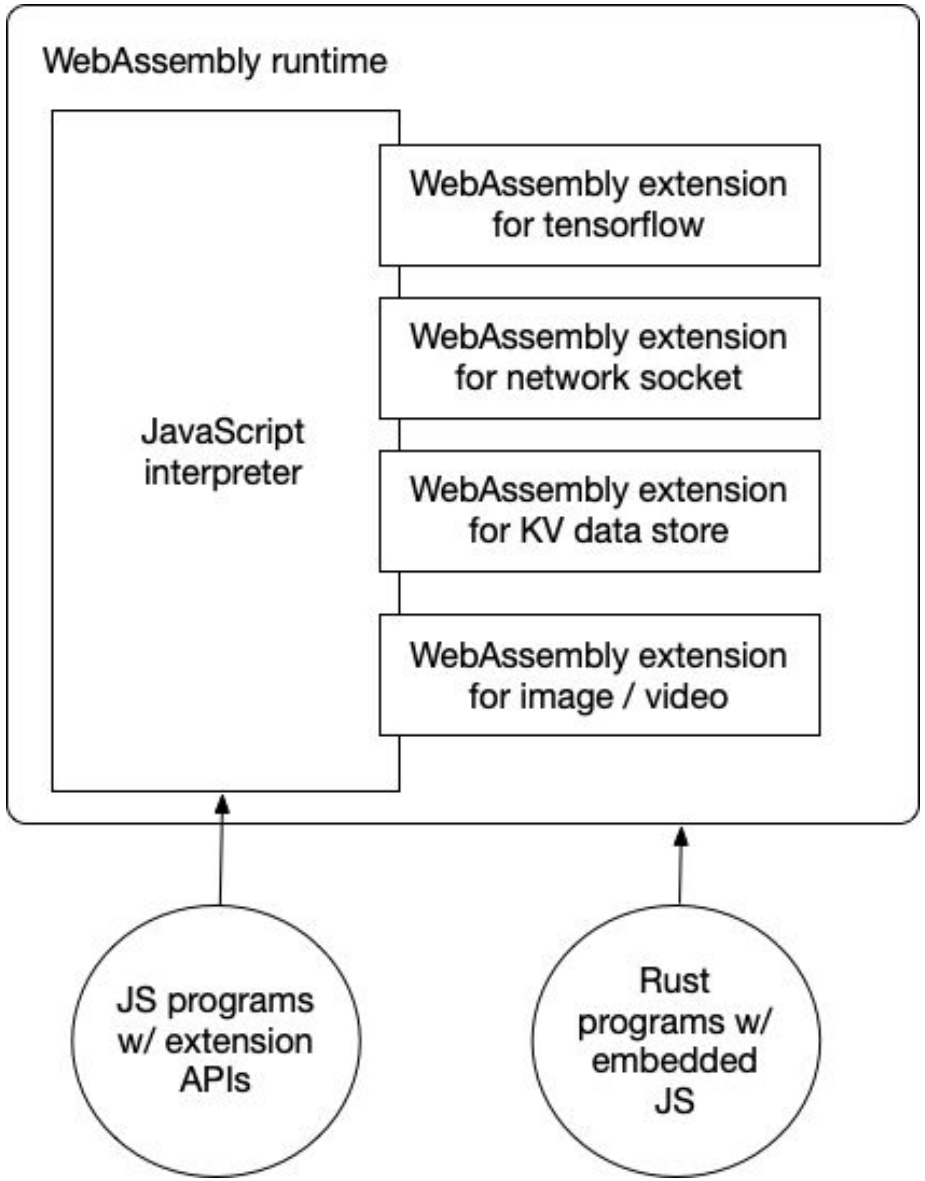
Rust

- Supports complex call parameters via wasmedge_bindgen
- Supports host networking via wasmedge_wasi_socket
- Supports a tokio-like async runtime
  - tokio MIO
  - hyper
  - reqwest
  - http_req
- Supports AI inference in Tensorflow, OpenVINO, and PyTorch
- Supports wasi-crypto -> rustls -> HTTPS
- Ongoing: SSR for Rust web frameworks? (e.g., Yew)

https://wasmedge.org/book/en/dev/rust.html

JavaScript – why

- Safer
  - Fine-grained security sandbox
  - Very small attack service
- Smaller footprint
- Probably faster
  - Apple to apple comp w same safety
  - Optimize via Rust implementations

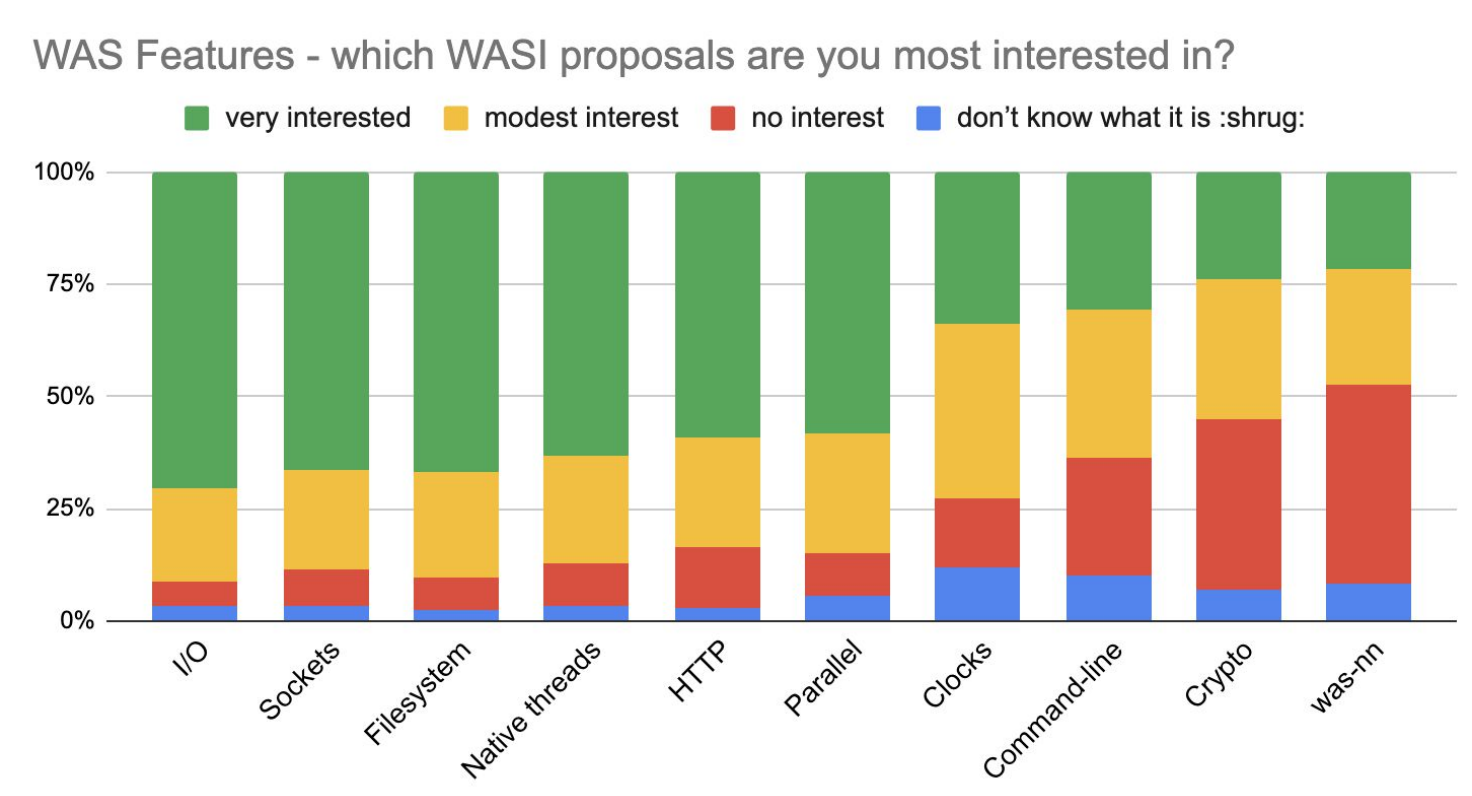


JavaScript – how

- Aims to support all Node.js APIs
  - Full support for HTTP / HTTPS networking
  - The fetch() API
- Supports AI inference
- Supports JS modules
  - ES6
  - CJS and NPM
- Supports React streaming SSR
- Supports JS APIs implemented in Rust!

WasmEdgeRuntime

Libraries and APIs support



HTTP service – for microservices

- Asynchronous & non-blocking socket APIs for concurrent requests
- Rust
  - tokio / mio
  - hyper
  - https://github.com/WasmEdge/wasmedge_hyper_demo/
- JavaScript
  - Node.js server API

```rust
#[tokio::main(flavor = "current_thread")]
async fn main() -> Result<(), Box<dyn std::error::Error + Send + Sync>> {
    let addr = SocketAddr::from(([0, 0, 0, 0], 8080));

    let listener = TcpListener::bind(addr).await?;
    println!("Listening on http://{}", addr);
    loop {
        let (stream, _) = listener.accept().await?;

        tokio::task::spawn(async move {
            if let Err(err) = Http::new().serve_connection(stream, service_fn(echo)).await {
                println!("Error serving connection: {:?}", err);
            }
        });
    }
}
```

The main server loop

The request handler

```rust
async fn echo(req: Request<Body>) -> Result<Response<Body>, hyper::Error> {
    match (req.method(), req.uri().path()) {
        // Serve some instructions at /
        (&Method::GET, "/") => Ok(Response::new(Body::from(
            "Try POSTing data to /echo such as: `curl localhost:3000/echo -XPOST -d 'hello world'`",
        ))),

        // Simply echo the body back to the client.
        (&Method::POST, "/echo") => Ok(Response::new(req.into_body())),

        (&Method::POST, "/echo/reversed") => {
            let whole_body = hyper::body::to_bytes(req.into_body()).await?;

            let reversed_body = whole_body.iter().rev().cloned().collect::<Vec<u8>>();
            Ok(Response::new(Body::from(reversed_body)))
        }

        // Return the 404 Not Found for other routes.
        _ => {
            let mut not_found = Response::default();
            *not_found.status_mut() = StatusCode::NOT_FOUND;
            Ok(not_found)
        }
    }
}
```

https://github.com/WasmEdge/wasmedge_hyper_demo

Web service clients

- Asynchronous & non-blocking socket APIs for concurrent requests
- Both HTTP and HTTPS are supported
- Rust
  - tokio / mio
  - reqwest: https://github.com/WasmEdge/wasmedge_reqwest_demo
  - http_req: https://github.com/second-state/http_req
- JavaScript
  - The fetch() API

```rust
let url = "http://eu.httpbin.org/get?msg=WasmEdge";

eprintln!("Fetching {:?}...", url);

let res = reqwest::get(url).await?;

eprintln!("Response: {:?} {}", res.version(), res.status());
eprintln!("Headers: {:#?}\n", res.headers());

let body = res.text().await?;
println!("GET: {}", body);
```

```rust
let client = reqwest::Client::new();

let res = client
    .post("http://eu.httpbin.org/post")
    .body("msg=WasmEdge")
    .send()
    .await?;
let body = res.text().await?;

println!("POST: {}", body);
```

https://github.com/WasmEdge/wasmedge_reqwest_demo/

Database clients

- Asynchronous & non-blocking socket APIs for concurrent requests
- Supports all MySQL compatible databases (e.g., MySQL, MariaDB, TiDB, TDEngine etc.)
- Supports anna-rs: a “coordination free” KVS for any scale
- Rust
  - MySQL: https://github.com/WasmEdge/wasmedge-db-examples/
  - anna-rs: https://github.com/WasmEdge/wasmedge-db-examples/tree/main/anna

```rust
let orders = vec![
    Order::new(1, 12, 2, 56.0, 15.0, 2.0, String::from("Mataderos 2312")),
    Order::new(2, 15, 3, 256.0, 30.0, 16.0, String::from("1234 NW Bobcat")),
    Order::new(3, 11, 5, 536.0, 50.0, 24.0, String::from("20 Havelock")),
    Order::new(4, 8, 8, 126.0, 20.0, 12.0, String::from("224 Pandan Loop")),
    Order::new(5, 24, 1, 46.0, 10.0, 2.0, String::from("No.10 Jalan Besar")),
];

r"INSERT INTO orders (order_id, production_id, quantity, amount, shipping, tax, shipping_address)
  VALUES (:order_id, :production_id, :quantity, :amount, :shipping, :tax, :shipping_address)"
    .with(orders.iter().map(|order| {
        params! {
            "order_id" => order.order_id,
            "production_id" => order.production_id,
            "quantity" => order.quantity,
            "amount" => order.amount,
            "shipping" => order.shipping,
            "tax" => order.tax,
            "shipping_address" => &order.shipping_address,
        }
    }))
    .batch(&mut conn)
    .await?;
```

Create

```rust
let loaded_orders = "SELECT * FROM orders"
    .with(())
    .map(
        &mut conn,
        |(order_id, production_id, quantity, amount, shipping, tax, shipping_address)| {
            Order::new(
                order_id,
                production_id,
                quantity,
                amount,
                shipping,
                tax,
                shipping_address,
            )
        },
    )
    .await?;
dbg!(loaded_orders.len());
dbg!(loaded_orders);
```

Query

https://github.com/WasmEdge/wasmedge-db-examples

A complete example

Lightweight and secure microservice with a database backend

In this repo, we demonstrate a microservice written in Rust, and connected to a MySQL database. It supports CURD operations on a database table via a HTTP service interface. The microservice is compiled into WebAssembly and runs in the WasmEdge Runtime, which is a secure and lightweight alternative to natively compiled Rust apps in Linux containers. The WasmEdge Runtime can be managed and orchestrated by container tools such as the Docker CLI, Podman, as well as almost all flavors of Kubernetes. It also works with microservice management frameworks such as Dapr.

> Everything described in this document is captured in the GitHub Actions CI workflow.

https://github.com/second-state/microservice-rust-mysql

AI inference

- Full native access to AI frameworks and underlying hardware (e.g., GPUs and TPUs)
  - PyTorch
  - Tensorflow
  - OpenVINO
- Rust: https://wasmedge.org/book/en/write_wasm/rust/wasinn.html
- JavaScript: https://wasmedge.org/book/en/write_wasm/js/tensorflow.html

```rust
(&Method::POST, "/classify") => {
    let buf = hyper::body::to_bytes(req.into_body()).await?;
    let flat_img = wasmedge_tensorflow_interface::load_jpg_image_to_rgb8(&buf, 224, 224);

    let mut session = wasmedge_tensorflow_interface::Session::new(&model_data, wasmedge_tensorflow_interface::ModelType::TensorFlowLite);
    session.add_input("input", &flat_img, &[1, 224, 224, 3])
        .run();
    let res_vec: Vec<u8> = session.get_output("MobilenetV1/Predictions/Reshape_1");

    let mut i = 0;
    let mut max_index: i32 = -1;
    let mut max_value: u8 = 0;
    while i < res_vec.len() {
        let cur = res_vec[i];
        if cur > max_value {
            max_value = cur;
            max_index = i as i32;
        }
        i += 1;
    }

    let mut label_lines = labels.lines();
    for _i in 0..max_index {
      label_lines.next();
    }
    let class_name = label_lines.next().unwrap().to_string();

    Ok(Response::new(Body::from(format!("{} is detected with {}/255 confidence", class_name, max_value))))
}
```

Server-side rendering (SSR)

isomorphic.secondstate.io

Server-side rendering on WasmEdge

Github Repository

https://github.com/second-state/create-react-app-ssr-exa

Framework Preset

Create React App | Percy

Build Script — Override

```
yarn
yarn build
yarn build-server
```

Run Command — Override

wasmedge | --dir .:. wasmedge_quickjs.wasm server-buil

Deployed successfully on
http://isomorphic.secondstate.io:60009

Deploy

```
The project was built assuming it is hosted at /.
You can control this with the homepage field in your package.json.

The build folder is ready to be deployed.
You may serve it with a static server:

  yarn global add serve
  serve -s build

Find out more about deployment here:

  https://cra.link/deployment

Done in 9.75s.
yarn run v1.22.18
warning Skipping preferred cache folder "/home/ssr/.cache/yarn" because i
warning Selected the next writable cache folder in the list, will be "/tm
$ NODE_ENV=development webpack --config webpack-wasmedge.server.js --mode
warning Cannot find a suitable global folder. Tried these: "/usr/local, /
asset index.js 258 KiB [emitted] [javascript module] (name: main)
runtime modules 937 bytes 4 modules
cacheable modules 240 KiB
  modules by path ./node_modules/ 232 KiB 7 modules
  modules by path ./src/ 4.65 KiB
    ./src/App.js 899 bytes [built] [code generated]
    ./src/logo.svg 2.64 KiB [built] [code generated]
    ./src/App.css 1.14 KiB [built] [code generated]
  ./server/wasmedge.js 2.62 KiB [built] [code generated]
external "std" 42 bytes [built] [code generated]
external "wasi_http" 42 bytes [built] [code generated]
external "wasi_net" 42 bytes [built] [code generated]
external "process" 42 bytes [built] [code generated]
webpack 5.69.1 compiled successfully in 1805 ms
Done in 2.28s.

********************
wasmedge --dir .:. wasmedge_quickjs.wasm server-build/index.js
```

https://wasmedge.org/book/en/write_wasm/js/ssr.html

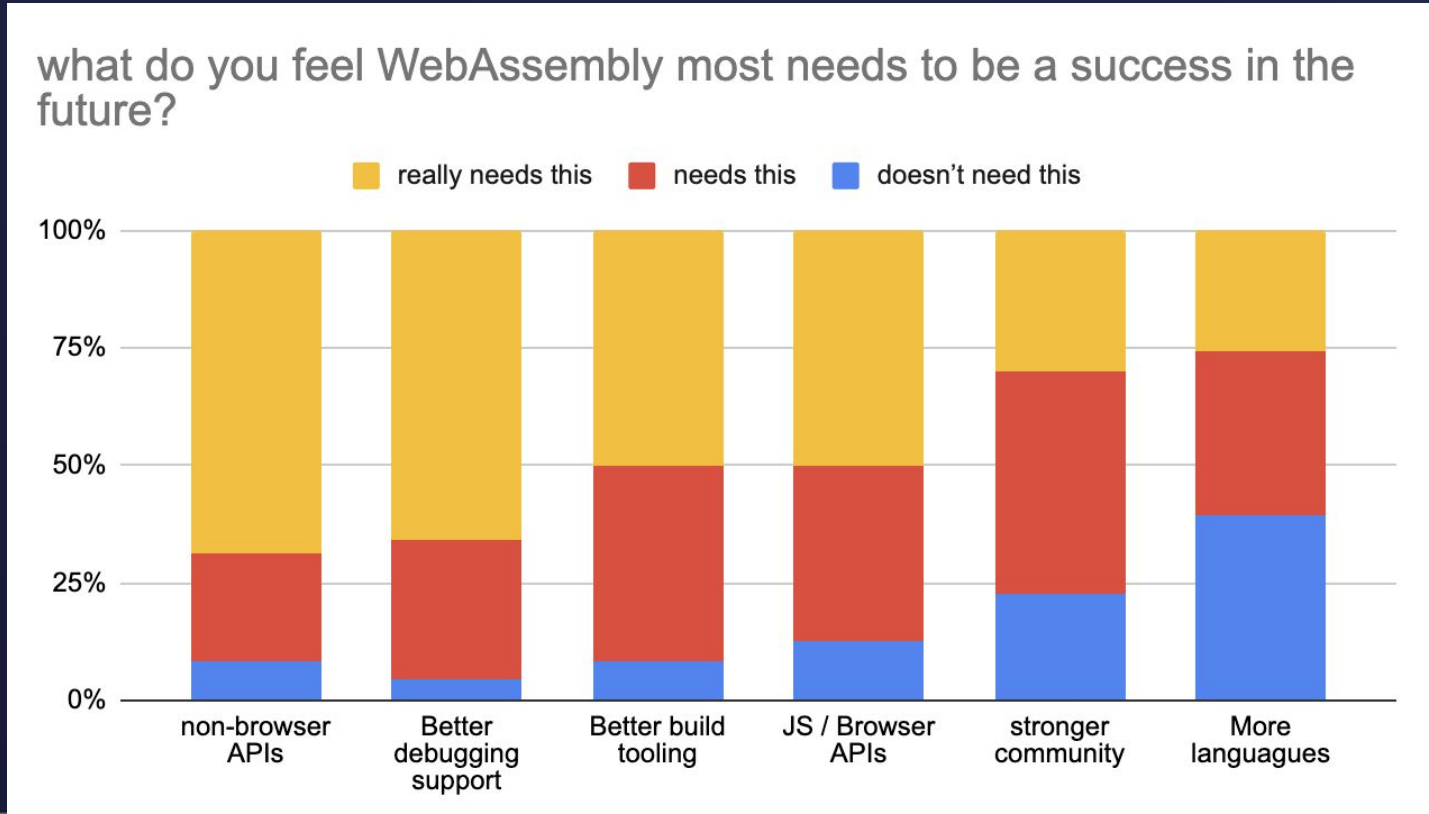
```
import * as React from 'react';
import { renderToPipeableStream } from 'react-dom/server';
import { createServer } from 'http';
import * as std from 'std';

import App from './component/App.js';
import { DataProvider } from './component/data.js'

let assets = {
  'main.js': '/main.js',
  'main.css': '/main.css',
};

const css = std.loadFile('./public/main.css')

function createServerData() {
  let done = false;
  let promise = null;
  return {
    read() {
      if (done) {
        return;
      }
      if (promise) {
        throw promise;
      }
      promise = new Promise(resolve => {
        setTimeout(() => {
          done = true;
          promise = null;
          resolve();
        }, 2000);
      });
      throw promise;
    },
  };
}
```

```
createServer((req, res) => {
  print(req.url)
  if (req.url == '/main.css') {
    res.setHeader('Content-Type', 'text/css; charset=utf-8')
    res.end(css)
  } else if (req.url == '/favicon.ico') {
    res.end()
  } else {
    res.setHeader('Content-type', 'text/html');

    res.on('error', (e) => {
      print('res error', e)
    })
    let data = createServerData()
    print('createServerData')

    const stream = renderToPipeableStream(
      <DataProvider data={data}>
        <App assets={assets} />
      </DataProvider>, {
      onShellReady: () => {
        stream.pipe(res)
      },
      onShellError: (e) => {
        print('onShellError:', e)
      }
    }
    );
  }
}).listen(8002, () => {
  print('listen 8002...')
})
```

https://wasmedge.org/book/en/write_wasm/js/ssr.html

WasmEdgeRuntime

Developer tooling



Docker

- Integrated support for WasmEdge in
  - Docker CLI
  - Docker Compose

https://github.com/second-state/microservice-rust-mysql

```
FROM scratch
ENTRYPOINT [ "order_demo_service.wasm" ]
```

```yaml
services:
  server:
    image: server
    build:
      context: .
      platforms:
        - wasi/wasm32
    ports:
      - 8080:8080
    network_mode: host
    environment:
      DATABASE_URL: mysql://root:pass@localhost:3306/mysql
      RUST_BACKTRACE: full
    runtime: io.containerd.wasmedge.v1
  db:
    image: mysql:8
    environment:
      MYSQL_ROOT_PASSWORD: pass
    network_mode: host
```

Podman

- Use Podman to manage WasmEdge applications
- Developed by Liquid Reply

How it works

Podman machine creates a CoreOS QEMU VM to run podman. CoreOS has podman already installed, it uses containerd and crun to run OCI containers. Crun can also run WebAssembly but it needs to be enabled during compiletime. Therefore we build a version of crun with WasmEdge as WebAssembly runtime and put it together with the WasmEdge libs in the VM image for podman machine.

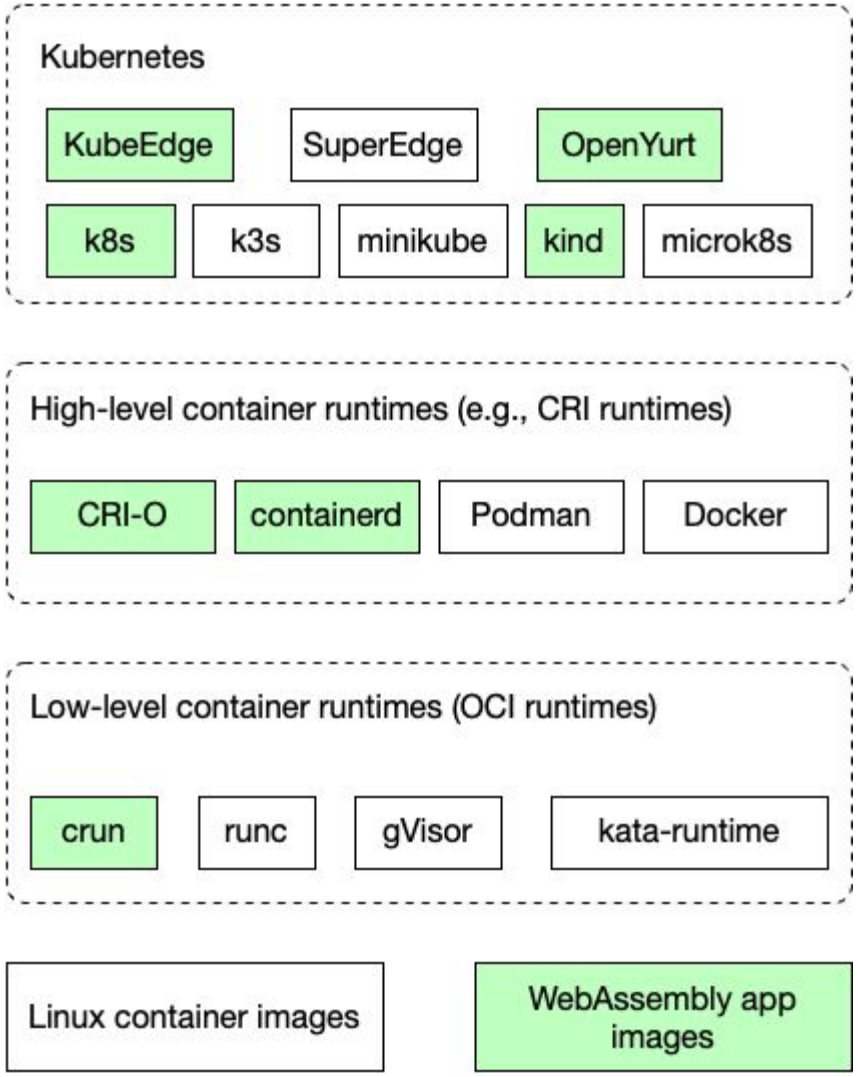
Crun uses annotations to distinguish between standard linux and wasm-containers. The annotations are `module.wasm.image/variant=compat` or `run.oci.handler=wasm` when running a wasm-container with podman it always need an annotation like this `--annotation run.oci.handler=wasm` (see example).

https://github.com/KWasm/podman-wasm

Kubernetes

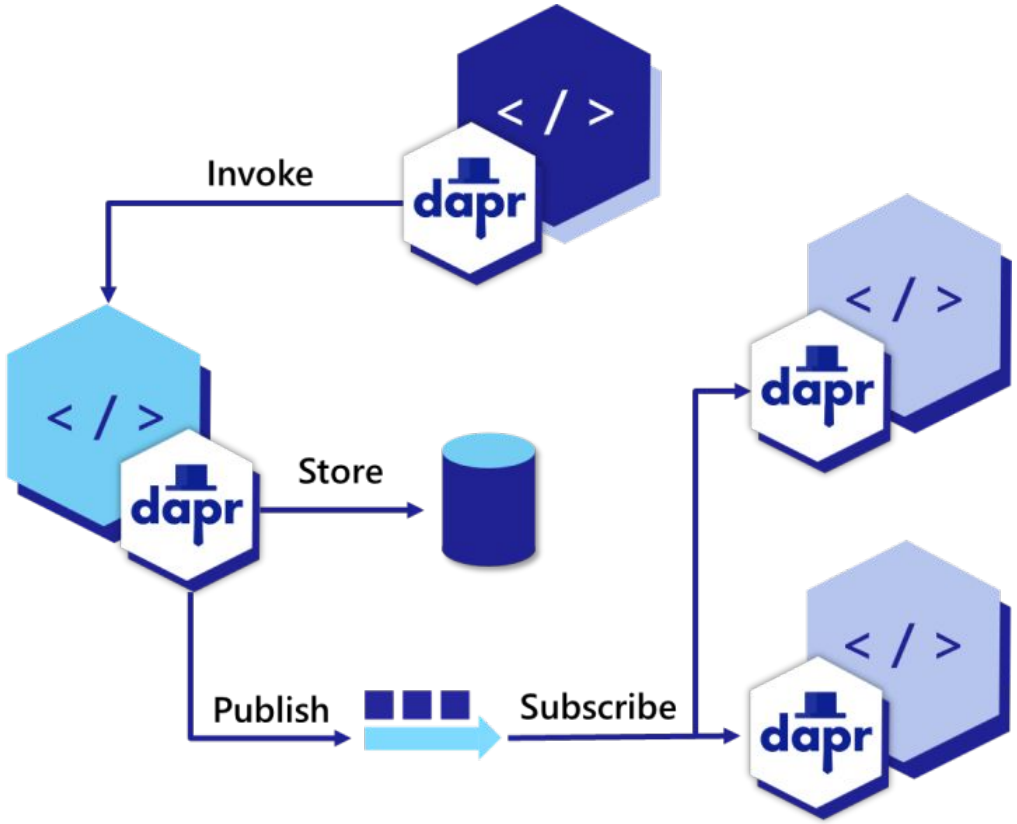
- Transparently supports containers and WebAssembly apps from the same tool and in the same cluster
- Working demos for
  - Kubernetes, Kind, OpenYurt, KubeEdge, OpenEdge
  - CRI-O, containerd

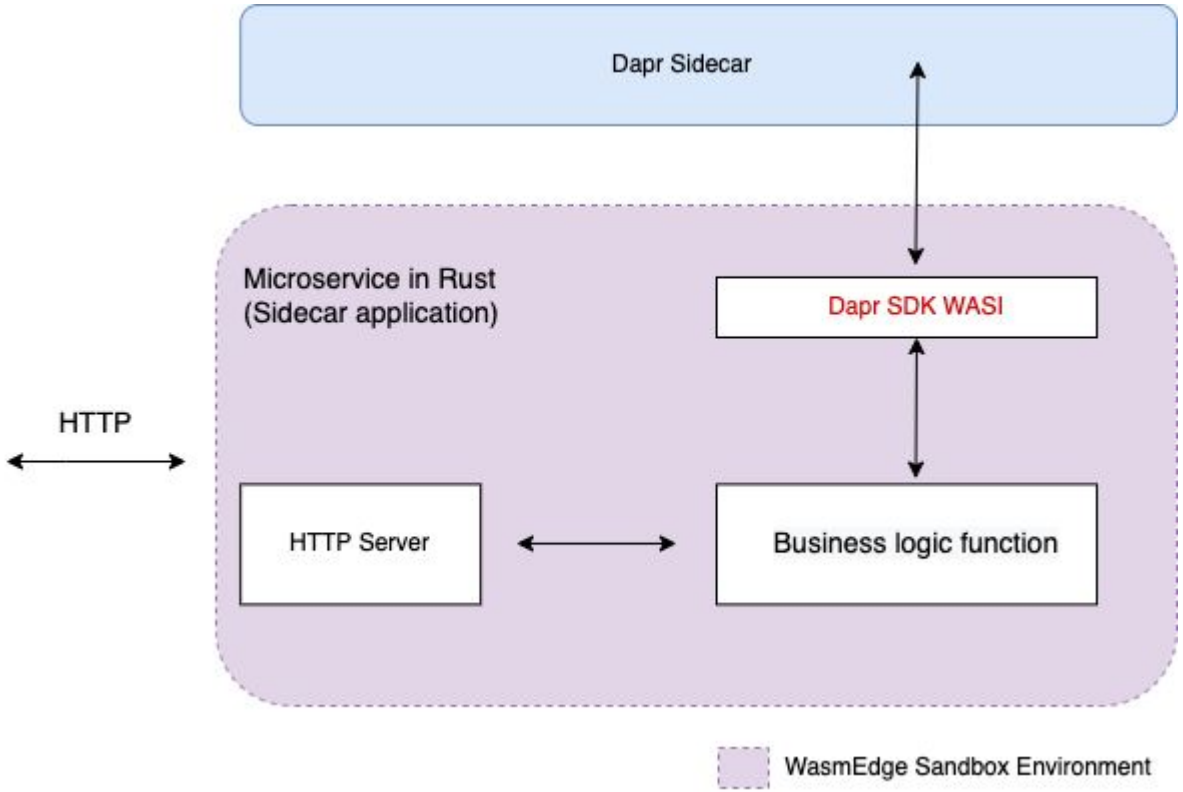
https://wasmedge.org/book/en/use_cases/kubernetes.html



The container ecosystem

Dapr





Example app: https://github.com/second-state/dapr-wasm

Dapr SDK for Wasm: https://github.com/second-state/dapr-sdk-wasi

- Service discovery and invocation
- State management

```rust
// Connect to the attached sidecar
let client = dapr::Dapr::new(3503);
let ts = Utc::now().timestamp_millis();

let kvs = json!({
    "event_ts": ts,
    "op_type": "grayscale",
    "input_size": image_data.len()
});
client.invoke_service("events-service", "create_event", kvs).await?;

let kvs = json!([{
    "key": ip, "value": ts
}]);
println!("KVS is {}", serde_json::to_string(&kvs)?);
client.save_state("statestore", kvs).await?;
```

- Secret management

```rust
let v = client.get_secret("local-store", "DB_URL:MYSQL").await?;
println!("MYSQL value is {}", v);
let db_url = v["DB_URL:MYSQL"].as_str().unwrap();
println!("Connection is {}", db_url);
```

https://github.com/second-state/dapr-wasm

Cloud-native WebAssembly

KubeCon | CloudNativeCon
North America 2022
BUILDING FOR THE ROAD AHEAD
DETROIT 2022

KubeCon | CloudNativeCon
North America 2022
BUILDING FOR THE ROAD AHEAD
DETROIT 2022

October 24-28, 2021

WasmEdge brings cloud-native tooling to WebAssembly (e.g., Docker & K8s)

https://hackmd.io/@wasmedge/SJCmYPDmo

Create a database-driven HTTP microservice in WebAssembly

https://github.com/second-state/microservice-rust-mysql

Develop a Dapr-based service mesh in WebAssembly

https://github.com/second-state/dapr-wasm