Database apps in WebAssembly

WasmEdgeRuntime

Michael Yuan, WasmEdge Maintainer
https://github.com/WasmEdge/WasmEdge

Most server-side frameworks' first killer app is to babysit a database.

Think the Java Petstore and Ruby On Rails scaffolding.

SECOND STATE

Approaches for stateful WebAssembly functions

1. Use high-level host functions (many Wasm-based frameworks)
2. Embed Wasm in the database! (libsql, Single Store, Nebula Graph etc)
3. Reuse existing database clients and libraries
4. Reuse existing sidecar services

Wasm embedded in a database?

```
libsql > select classify(img_blob) from images where id = 1;
military uniform
```

https://github.com/libsql/libsql_bindgen/tree/master/examples/wasmedge

wasmedge ✓
@realwasmedge

The example allows users to store an image as a blob in a @libsqlhq database, and just use SQL to query what's on the image (image classification is done in WasmEdge using TensorflowLite). Check it out!

Michael Yuan @juntao · 4h

Love @sarna_dev talk on how to use Wasm UDFs in @libsqlhq at @wasm_io

Here are some unique UDFs WasmEdge @realwasmedge enables: HTTPS web services and AI inference inside the database! github.com/libsql/libsql_...

Just use a database client you already know

```rust
#[tokio::main(flavor = "current_thread")]
async fn main() -> Result<()> {
    let opts = Opts::from_url(&*get_url()).unwrap();
    let builder = OptsBuilder::from_opts(opts);
    let constraints = PoolConstraints::new(5, 10).unwrap();
    let pool_opts = PoolOpts::default().with_constraints(constraints);

    let pool = Pool::new(builder.pool_opts(pool_opts));
    let mut conn = pool.get_conn().await.unwrap();
```

```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?;
```

```rust
    // query data
    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);
```

mysql_async v0.31.3

Tokio based asynchronous MySql client library.

#database #mysql #asynchronous #async

Readme | 61 Versions | Dependencies | Dependents

Stats Overview

1,142,770 Downloads all time

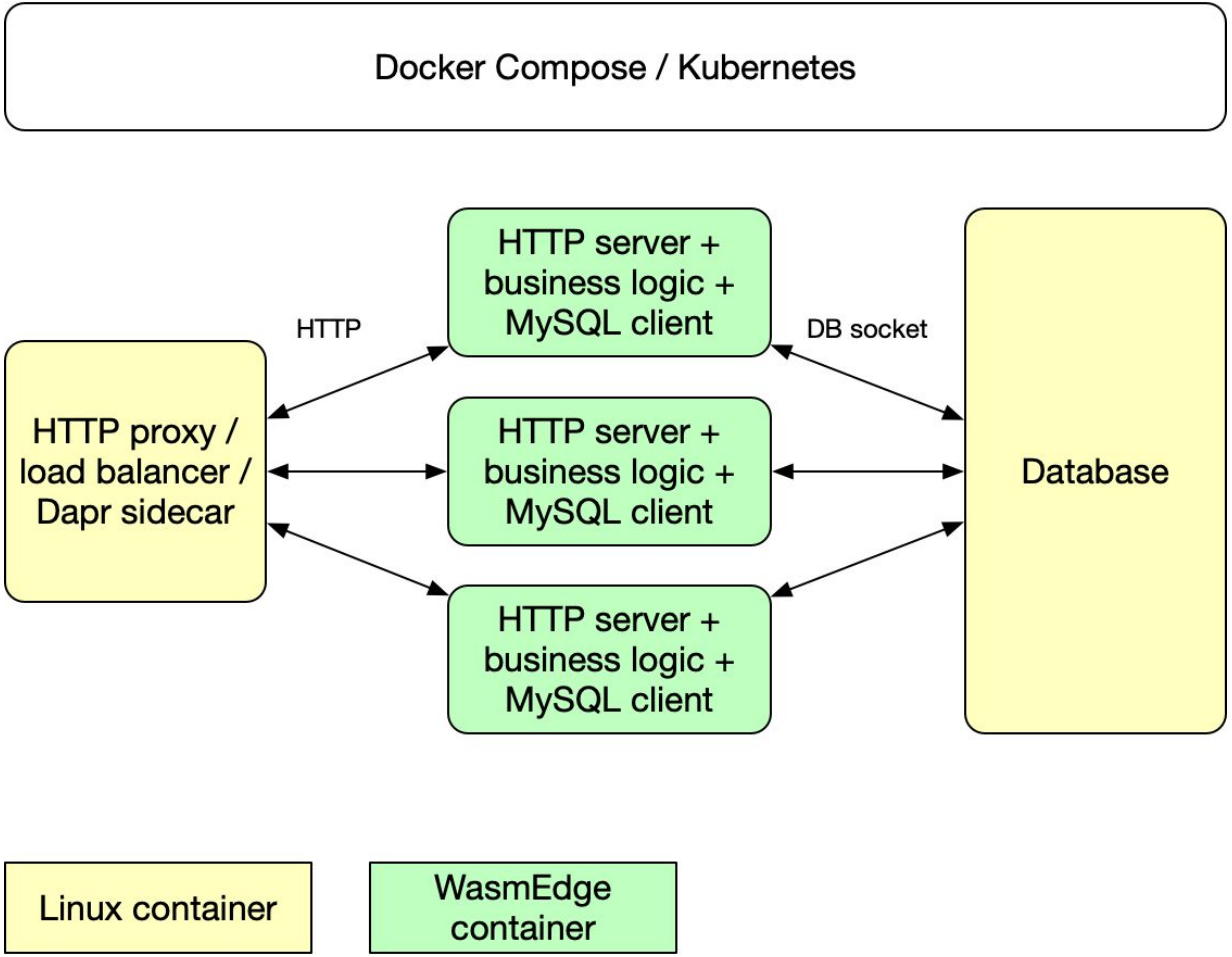
61 Versions published

www.secondstate.io

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

Demo: A database backed web service

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


Docker Compose / Kubernetes
HTTP server + business logic + MySQL client
HTTP
DB socket
HTTP proxy / load balancer / Dapr sidecar
HTTP server + business logic + MySQL client
Database
HTTP server + business logic + MySQL client
Linux container
WasmEdge container

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

SECOND STATE

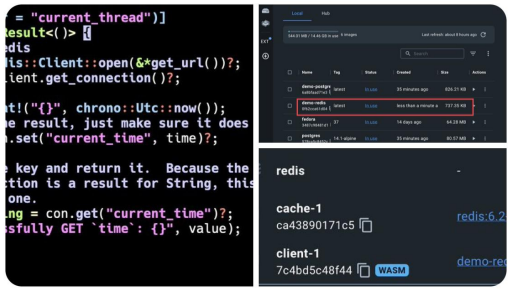
Light, fast and secure

wasmedge
@realwasmedge

A complete Redis app running inside a secure Wasm container managed by Docker + #Wasm. Total app size is 0.7MB and starts in milliseconds. (A comparable Linux container app for #redis is easily 50+MB).

github.com/WasmEdge/wasme...

@Redisinc @Docker



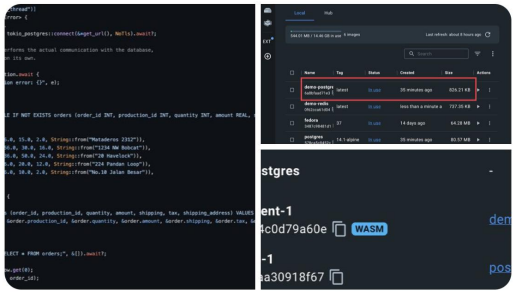
11:55 PM · 2/11/23 from Austin, TX · **12.2K** Views

wasmedge
@realwasmedge

A #PostgreSQL client app running inside a secure Wasm container managed by Docker + #Wasm. Total app size is 0.8MB. It runs anywhere and starts in milliseconds. (A comparable Linux container is easily 50MB).

github.com/WasmEdge/wasme...

@PostgreSQL @planetpostgres @Docker



6:50 PM · 2/14/23 from San Francisco, CA · **16K** Views

How it works

- WasmEdge WASI sockets
  - Support non-blocking sockets – crucial for data-intensive apps. It can handle multiple HTTP requests and associated database queries concurrently.
  - Support DNS
  - Support TLS
  - Support domain sockets
  - Also compatible with the simpler WASI-socket spec
- Guest app SDKs
  - Fork tokio and MIO to add WasmEdge WASI target support
  - Maintain a tree of forks of database clients based on tokio_wasi
  - Create a Rust / Wasm SDK for Dapr API
  - Incorporate Rust functions to WasmEdge-QuickJS

SECOND STATE

Rust tokio-based clients and JavaScript node.js clients both work

SECOND STATE

Databases supported

SQLx

anna-rs

SECOND STATE

Socket support allows us to go beyond databases

Libraries and frameworks

- Rust
  - Tokio
  - MIO
  - hyper
  - reqwest
  - Mysql_async, postgres, sqlx
  - rskafka
  - redis, anna-rs
  - Dapr
- JavaScript
  - Node
  - fetch()
  - React SSR

SECOND STATE

I want more!

- Dapr is a sidecar to provide common services to microservices
  - Eg. key-value store, relational database, noSQL datastores, secret vault, service discovery, observability, pub/sub queues, actors etc.
- Deployment
  - Dapr sidecar and microservice are deployed in two containers in a single pod
  - They communicate via gRPC or HTTP
  - External KVS and databases can be accessed through Dapr API
- Currently supports **26 data stores**!
  - https://docs.dapr.io/reference/components-reference/supported-state-stores/
- Dapr SDK for WasmEdge: https://github.com/second-state/dapr-sdk-wasi
- Example: https://github.com/second-state/dapr-wasm

A future of component models

- All WasmEdge host functions are plugins
  - Even WASI will be a plugin
  - Supports alternative WASI implementations, such as WasmEdge sockets
- Plugins can be described by WAT files and be compatible with the Component Model specification
- Check out WasmEdge's Compoment Model tooling: https://github.com/second-state/witc

One more thing

- ChatGPT enabled serverless functions
  - Receive events from anywhere on the web
  - Use live data to query ChatGPT
  - Execute the actions suggested by ChatGPT
  - *Similar to "ChatGPT Plugins"*
- Enabled by WasmEdge WASI sockets
  - Receive trigger events (e.g., incoming chat message) via HTTPS
  - Query OpenAI API via async HTTPS
  - Store and retrieve chat thread history in Redis
  - Execute actions in OpenAI responses via HTTPS

Examples with GPT4

A telegram bot

- Code:
  https://github.com/flows-network/telegram-gpt

A GitHub Issues bot

- Code:
  https://github.com/flows-network/chatgpt-github-app
- Tutorial:
  https://www.freecodecamp.org/news/create-a-serverless-chatgpt-app/

SECOND
STATE

Thanks !