
KubeCon
CloudNativeCon
Europe 2023
TiKV

KubeCon | CloudNativeCon

Europe 2023

Create and Deploy a Lightweight Microservice in WebAssembly

Tai Hung-Ying & Vivian Hu & Michael Yuan

WasmEdge

https://github.com/WasmEdge/WasmEdge

Agenda

- The rise of lightweight microservices
- Why is WebAssembly a good fit?
- The WasmEdge approach
- Tutorial #1: Create a complete 3-tiered microservice with Docker+Wasm
- Tutorial #2: The anatomy of the microservice
- Tutorial #3: Do more with Dapr
- Tutorial #4: An event driven microservice with ChatGPT and GitHub!

You can find all resources here:
https://github.com/second-state/kubecon-eu-2023

KubeCon | CloudNativeCon

Europe 2023

The rise of lightweight microservices

Elon Musk @elonmusk · Nov 14

Btw, I'd like to apologize for Twitter being super slow in many countries. App is doing >1000 poorly batched RPCs just to render a home timeline!

24.3K 14.7K 173.8K

Elon Musk @elonmusk · Nov 14

Replying to @sampullara

I was told ~1200 RPCs independently by several engineers at Twitter, which matches # of microservices. The ex-employee is wrong.

Same app in US takes ~2 secs to refresh (too long), but ~20 secs in India, due to bad batching/verbose comms. Actually useful data transferred is low.

604 861 8,229

Elon Musk @elonmusk · Nov 14

Replying to @elonmusk and @sampullara

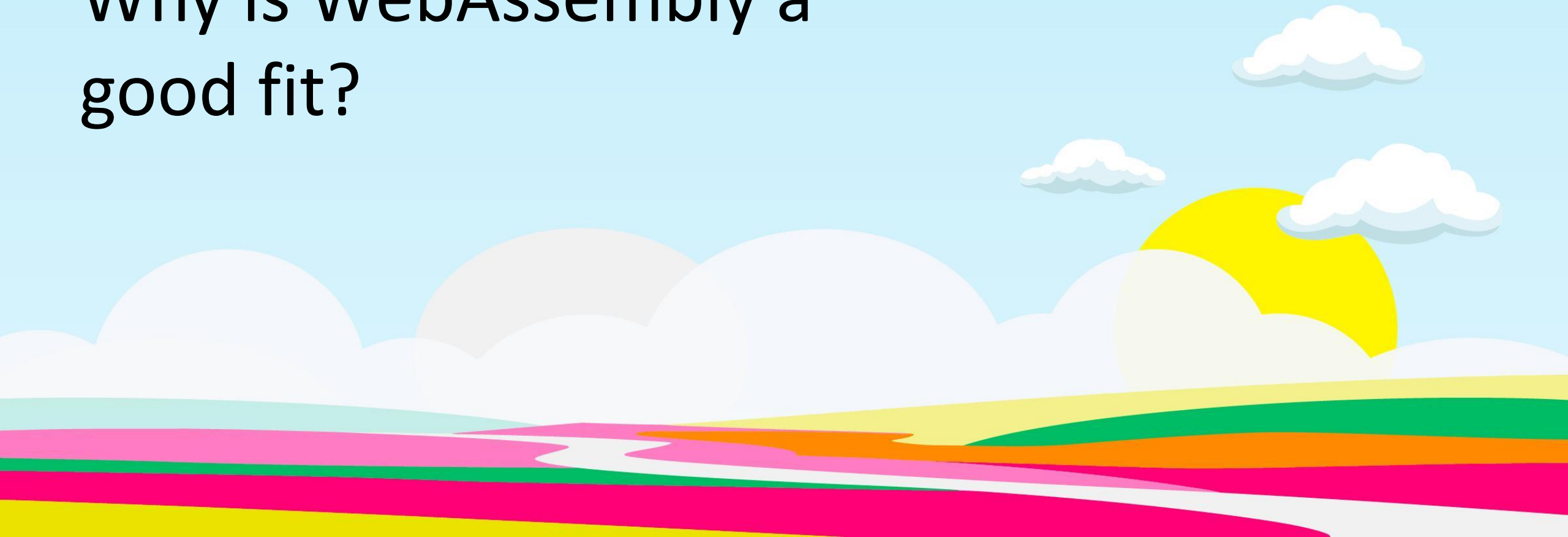
Part of today will be turning off the "microservices" bloatware. Less than 20% are actually needed for Twitter to work!

1,694 4,887 9,635

Pain points

- Heavyweight (container + Linux OS + frameworks + app)
- Slow, especially at startup time, requiring “warm up”
- A general trade-off between security and overhead
- Not portable – what happens with write-once-run-everywhere?

Why is WebAssembly a good fit?



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

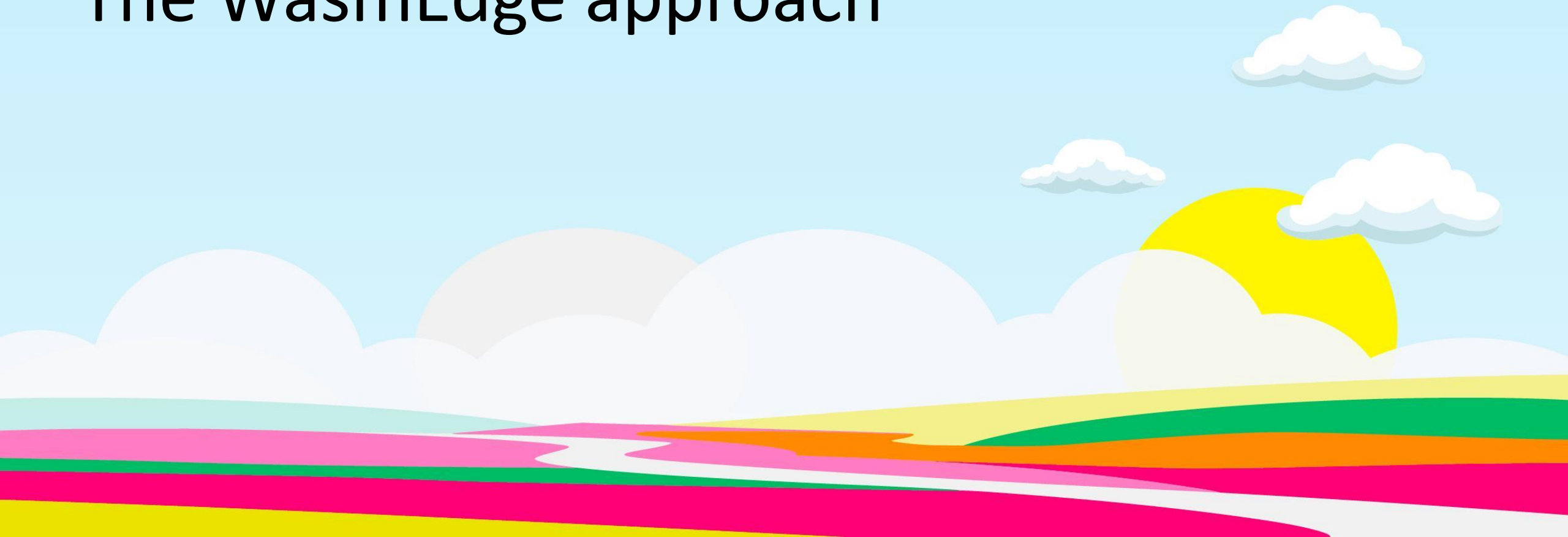
Too good to be true? There is no free lunch

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

Developer SDKs for Wasm microservices

- Spin by Fermyon – https://www.fermyon.com/spin
- wasmCloud by Cosmonic – https://wasmcloud.com/
- wasiCloud and component models – https://github.com/WebAssembly/WASI/issues/520
- Dapr SDK – https://github.com/second-state/dapr-sdk-wasmedge
- But what about existing microservices written in established frameworks such as tokio (for Rust) and node (for JavaScript)?

The WasmEdge approach



The WasmEdge runtime

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

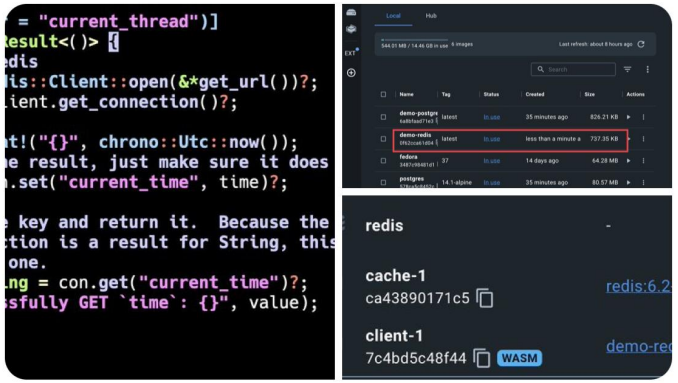
Light, fast and secure

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



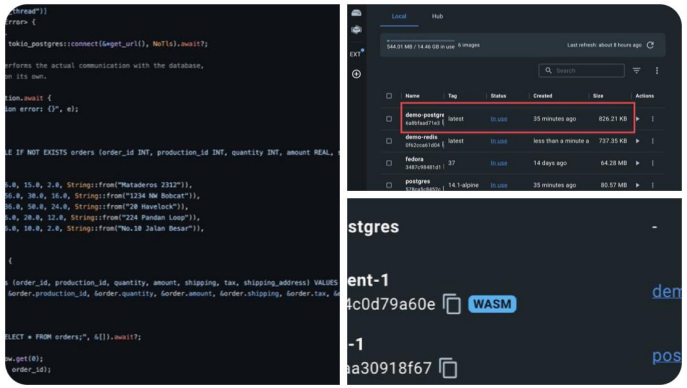
11:55 PM · 2/11/23 from Austin, TX · **12.2K** Views

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

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

Database supported

MySQL

MariaDB

PostgreSQL

SQLite

Microsoft SQL Server

redis

SQLx

anna-rs

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

Tooling

docker

kubernetes

containerd

podman

fedora

Red Hat

Demo Time

Tutorial #1: Create a complete 3-tiered microservice with Docker+Wasm

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

Tooling

Wasm is a fast, light alternative to Linux containers – try it out today with the Docker+Wasm Beta.

docker +WASM

Products Developers Pricing Blog About Us Partners

Develop faster. Run anywhere.

The most-loved Tool in Stack Overflow's 2022 Developer Survey.

Download Docker Desktop
Intel Chip

Apple Chip Windows Linux

WASM IS AWESOME

WHAT'S NEW

Docker + Wasm = Awesome!

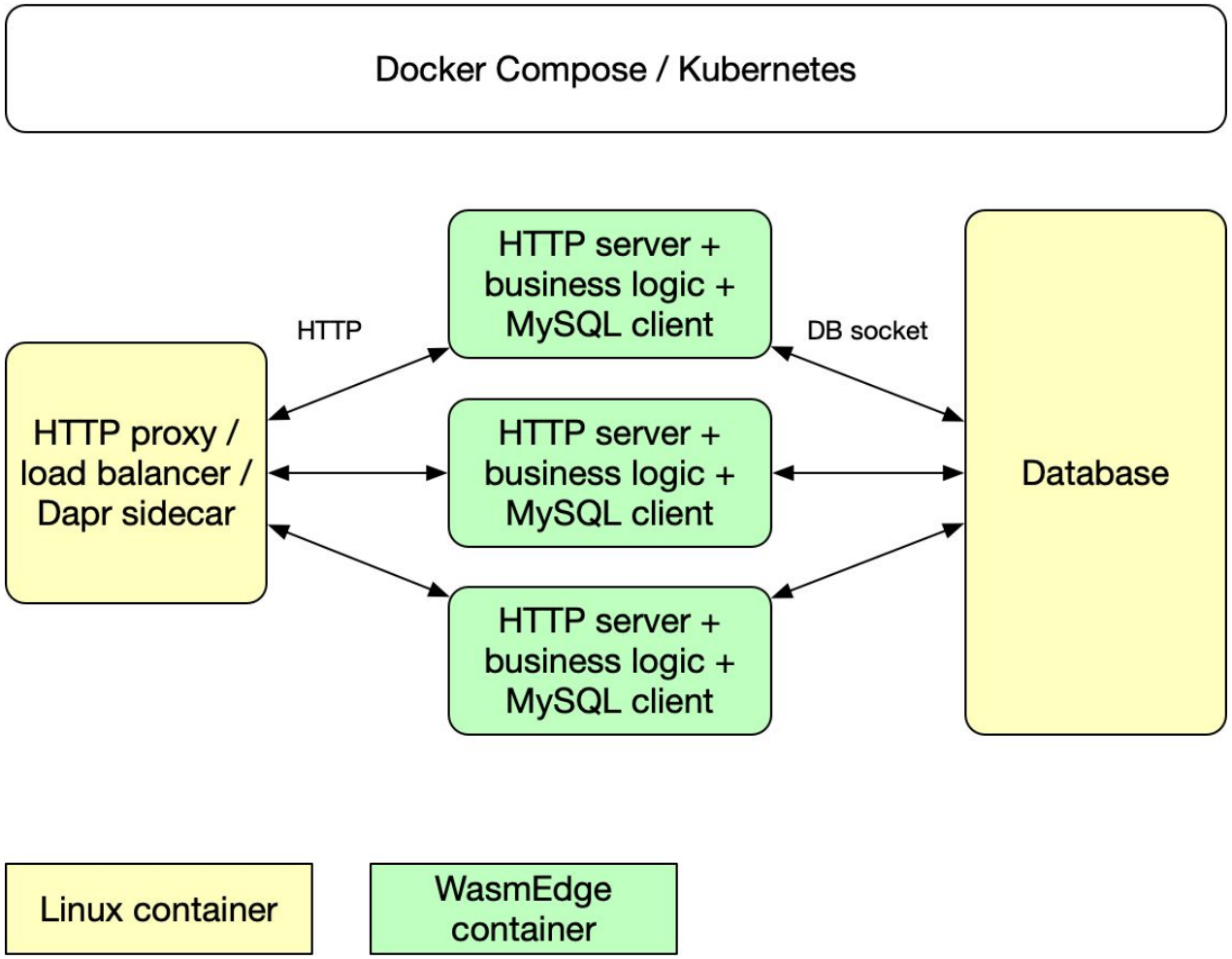
Wasm is a new, fast, and light alternative to the Linux/Windows containers you're using in Docker today — give it a try with the Docker+Wasm Beta.

→ Try it

What a typical microservice looks like

Linux container

With WasmEdge



a complete 3-tiered microservice

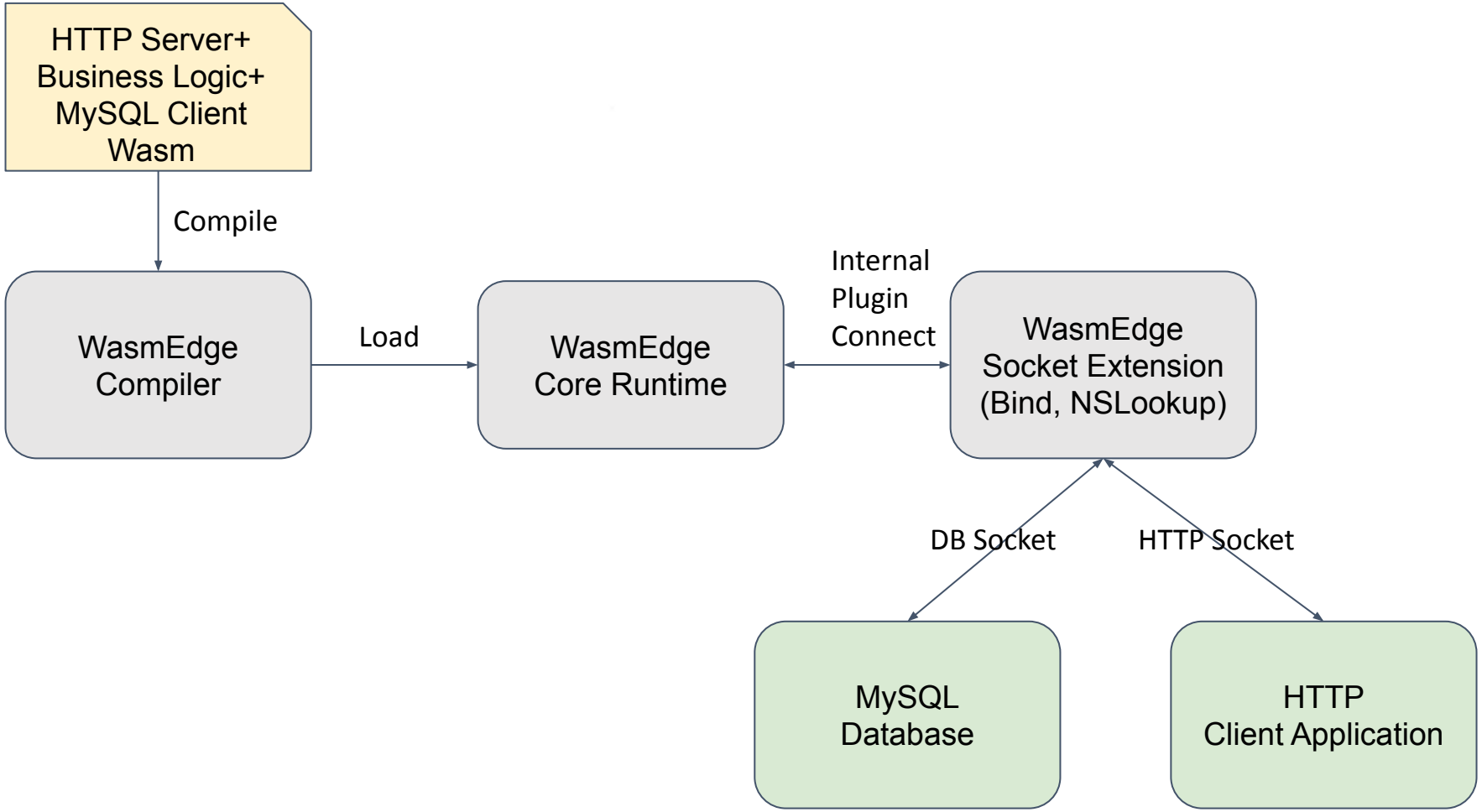
Steps to create a complete 3-tiered microservice with Docker+Wasm

1. Download the Docker Desktop v4.15 or above
2. Git clone the template project from GitHub: https://github.com/second-state/microservice-rust-mysql
3. Use a single command line *docker compose up* to build and run this template project
4. Open the URL *http://localhost:8090* in a browser and create a sample order

Tutorial #2: The anatomy of the microservice

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

Dig into how Wasm Runtime leverage the 3-tiered microservice



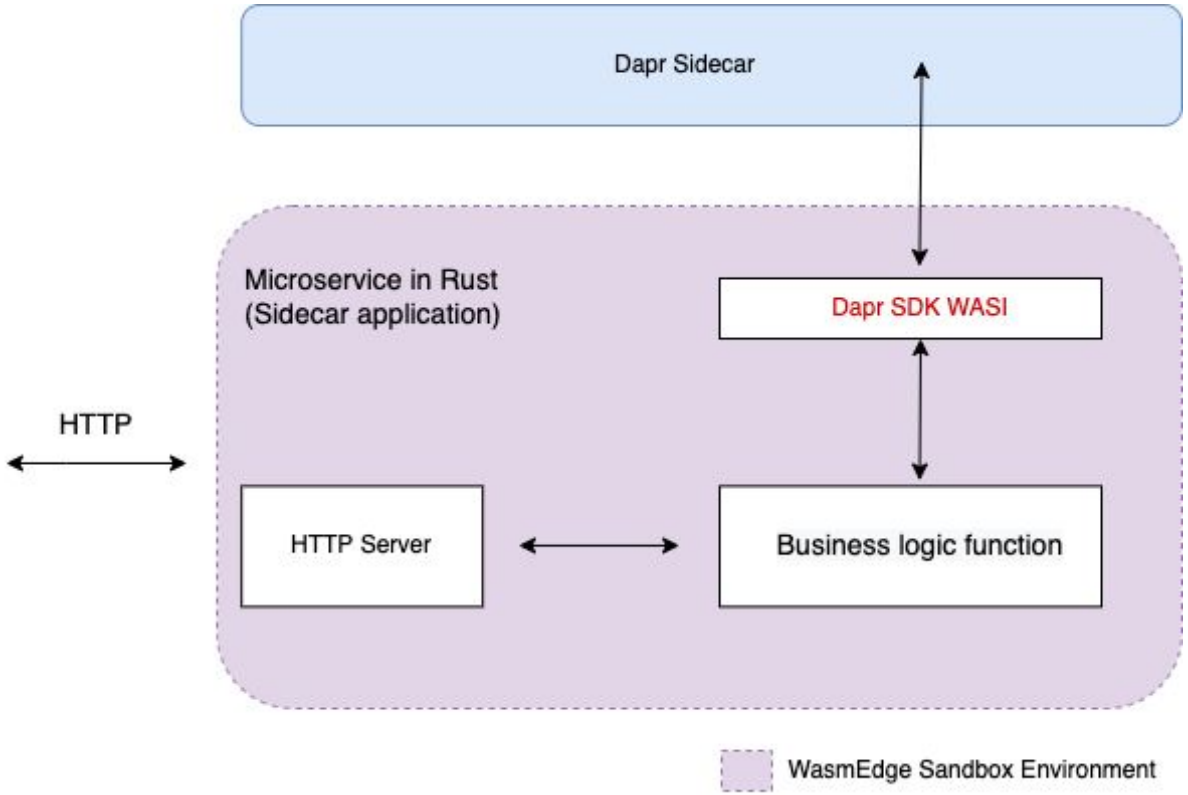
Steps to create a complete 3-tiered microservice with WasmEdge

1. Working environment (Linux)
   a. Install Rust Toolchain
   b. Install WasmEdge
   c. Install and Start the MySQL database
2. Build the application into a Wasm file
3. Use the Ahead-of-Time compilation to improve the performance
4. Execute the service engine with WasmEdge
5. Play with the service

Tutorial #3: Do more with Dapr

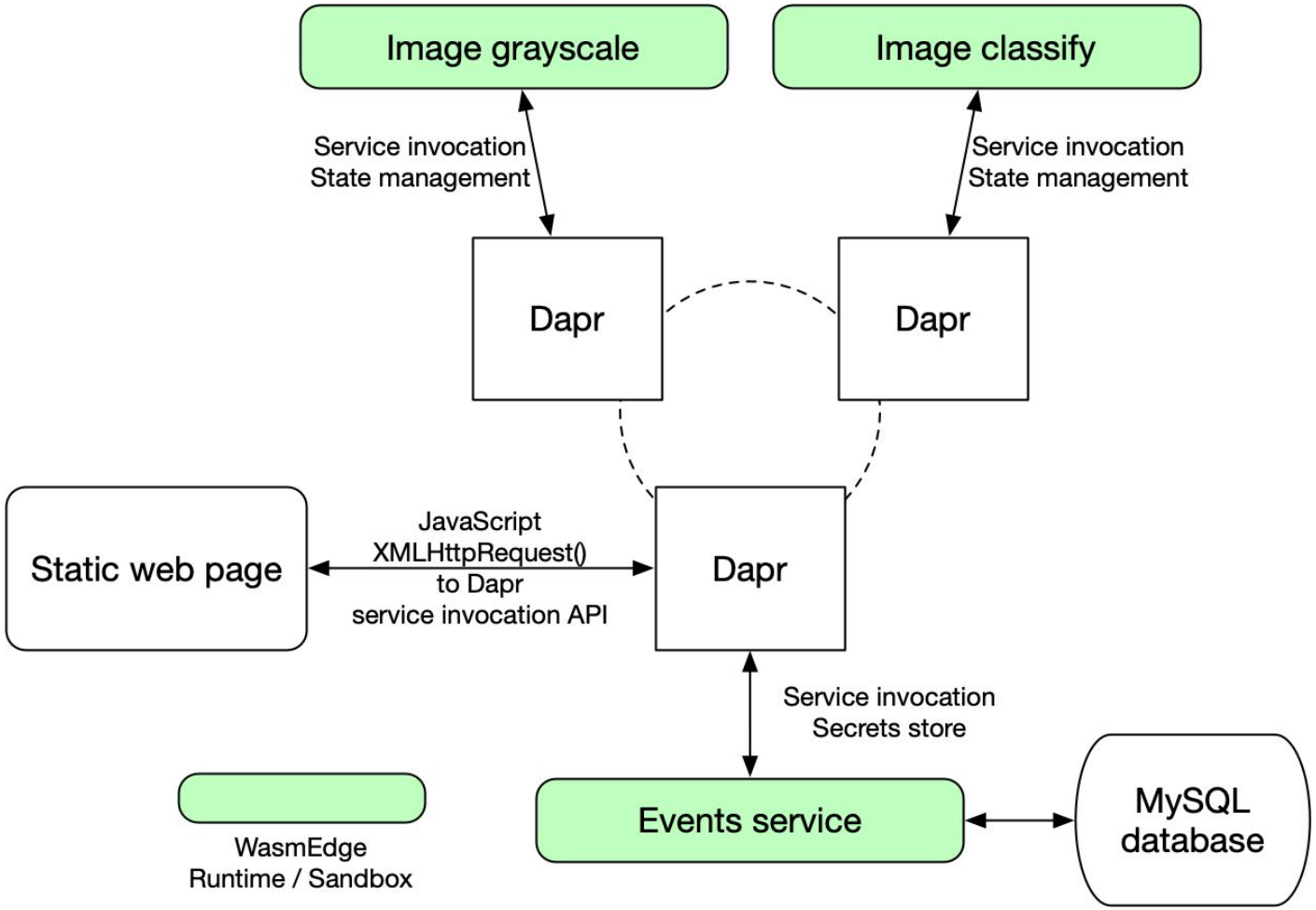
https://github.com/second-state/dapr-wasm

Introducing Dapr SDK for WASI



https://github.com/second-state/dapr-sdk-wasi

Architecture



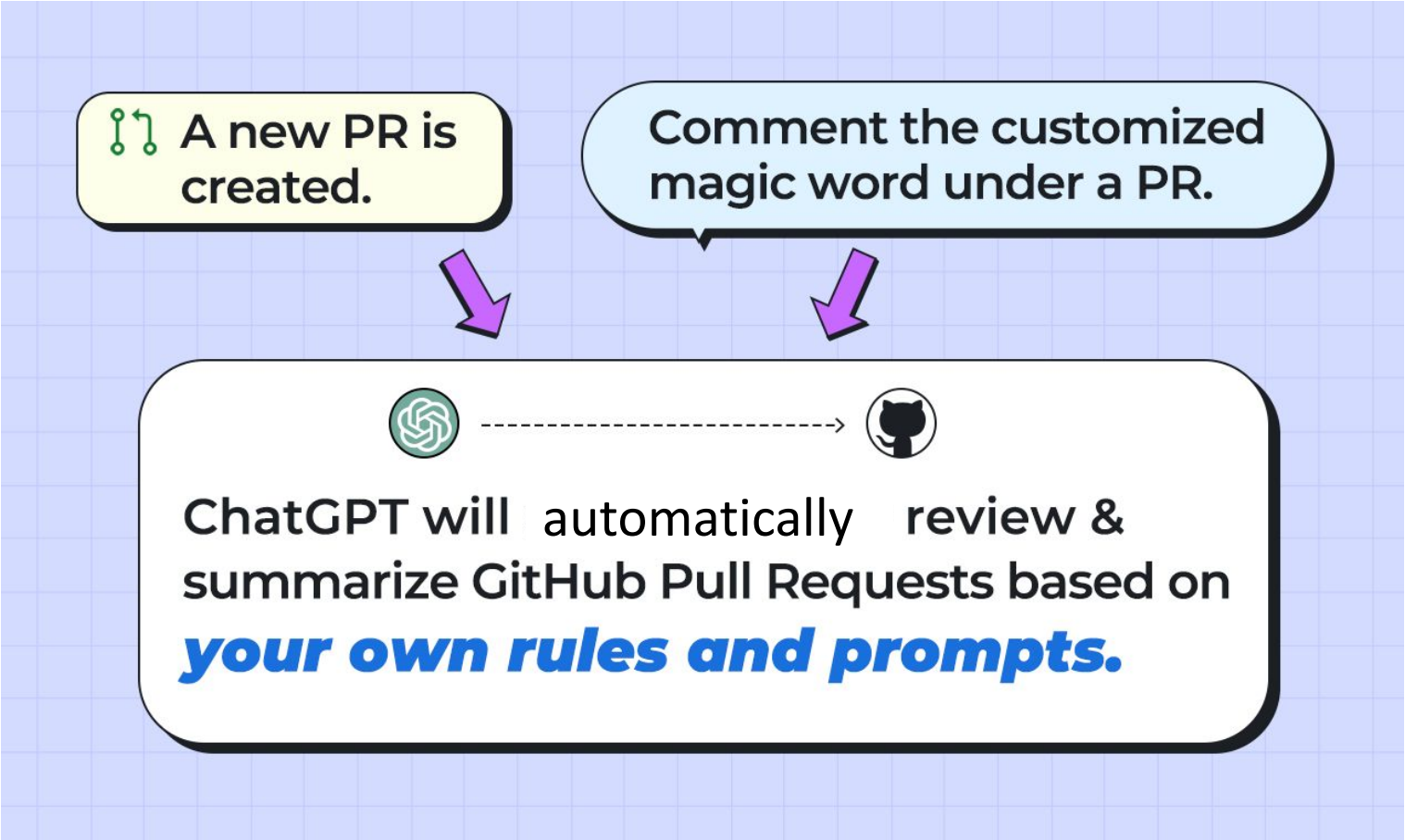
Steps to create microservices with Dapr

1. Prerequisites
   a. Install the Dapr CLI
   b. Install the WasmEdge Runtime
   c. Install Rust
   d. Install the MySQL or MariaDB or TiDB databases
2. Use the *dapr init* to start Docker
3. Git clone the template project: second-state/dapr-wasm
4. Build and run each microservice
5. Test the project

Tutorial #4: An event driven microservice with ChatGPT and GitHub!

https://github.com/flows-network/github-pr-summary/

- It’s a serverless way to build and deploy Wasm functions.
- All you need to do is to write your own business logic



Steps to create an event driven microservice with ChatGPT and GitHub!

1. Fork the github-pr-summary repo
2. Deploy your forked github repo to the flows.network
3. Connect the SaaS integrations you need
4. Test if this works

Resources

- The full tutorials: https://github.com/second-state/kubecon-eu-2023
- Join WasmEdge Discord server: https://discord.gg/U4B5sFTkFc
- CNCF #WasmEdge Slack Channel: https://slack.cncf.io/#wasmedge
- WasmEdge Twitter: https://twitter.com/realwasmedge
- WasmEdge GitHub repo: https://github.com/WasmEdge/WasmEdge
- flows.network: https://flows.network/

WasmEdge Discord


HELM
TiKV
KubeCon
CloudNativeCon
Europe 2023