



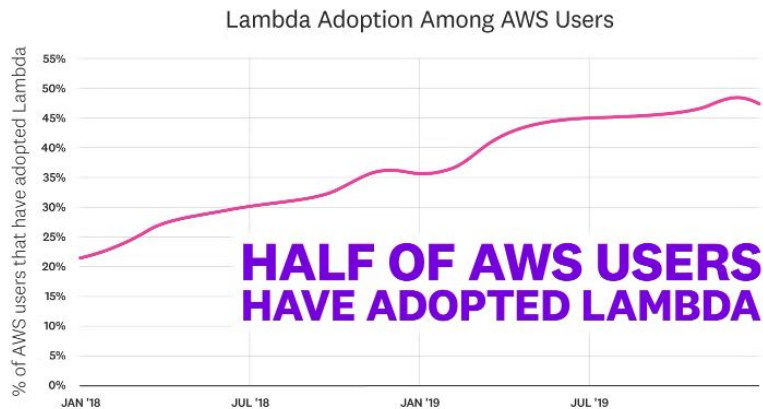
Serverless Functions on the Edge Cloud



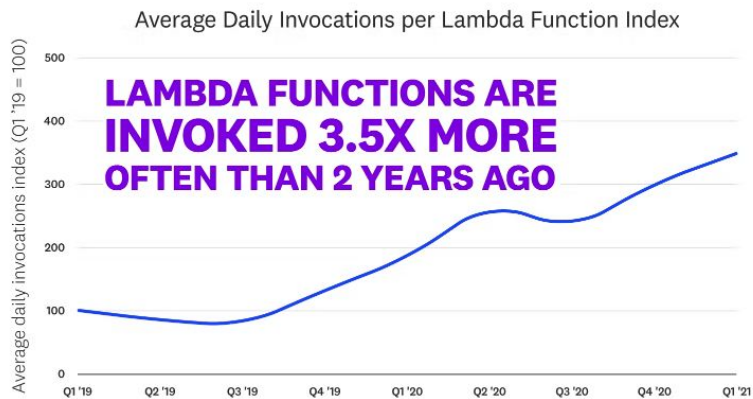
WasmEdgeRuntime

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

The rise of serverless functions



Source: Datadog



Source: Datadog

Two and half types of serverless functions

Serverless in the Public Cloud



Azure Functions



Google
Cloud
Functions



Tencent Serverless



阿里云

Serverless in the Edge Cloud

Vercel



火山引擎

netlify

fastly



amazon
cloudfront

CLLOUDFLARE

Akamai

What's the difference?

Serverless in the Public Cloud



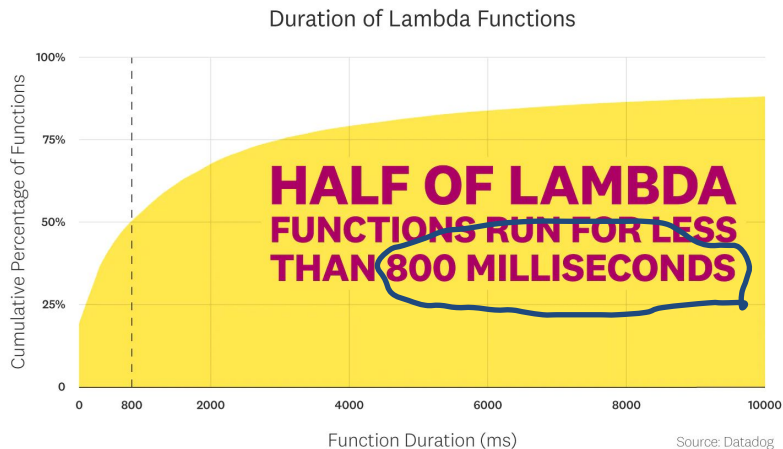
Serve the infrastructure
of a public cloud

Serverless in the Edge Cloud



Serve the application
outside of the infrastructure

Performance is a key requirement for edge clouds



Bundled Usage Model

Workers on the Bundled Usage Model are intended for use cases below **50 ms**. Bundled Workers limits are based on CPU time, rather than [duration](#). This means that the time limit does not include the time a Worker is waiting for responses from network calls. The billing model for Bundled Workers is based on requests that exceed the included number of requests on the Paid plan. Learn more about [Usage Model pricing](#).



Why WebAssembly

1

Secure for multi-tenancy cloud environments

2

Near native performance with sub-millisec cold start

3

Very small footprint (1/10 of LXC alternatives)

4

Large ecosystem (languages, SDKs, toolchains and standards)

https://wasmedge.org/wasm_linux_container/



WasmEdgeRuntime



CLOUD NATIVE
COMPUTING FOUNDATION

A high performance Wasm runtime

- Near native performance with LLVM-based AOT. Peer reviewed benchmark paper on IEEE Computer: <https://arxiv.org/abs/2010.07115>
- Supports a wide variety of OSES including seL4 RTOS, Open Harmony, OpenWRT, and others
- Supports all popular CPU architectures including Intel, ARM, Apple, and RISC-V

<https://github.com/WasmEdge/WasmEdge>

Optimized for cloud-native & edge

- Works seamlessly with the container ecosystem: Docker, containerd, CRI-O, various k8s flavors etc.
- Async networking with Tokio. Supports microservices, web service clients, database clients, cache, messaging queues etc.
- Works well with service frameworks such as k8s SDK, Dapr SDK etc.
- Native support (i.e., GPU) for AI inference with Tensorflow, OpenVINO, PyTorch etc.
- First class support for JavaScript, including full nodejs API, NPM, ES6, React SSR etc.

There is no free lunch

The trade-offs between a general computing environment and opinionated high performance frameworks

Serverless in the Public Cloud

Just use Linux



Serverless in the Edge Cloud

Opinionated languages and frameworks



Tooling ecosystem

- 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>

- 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!

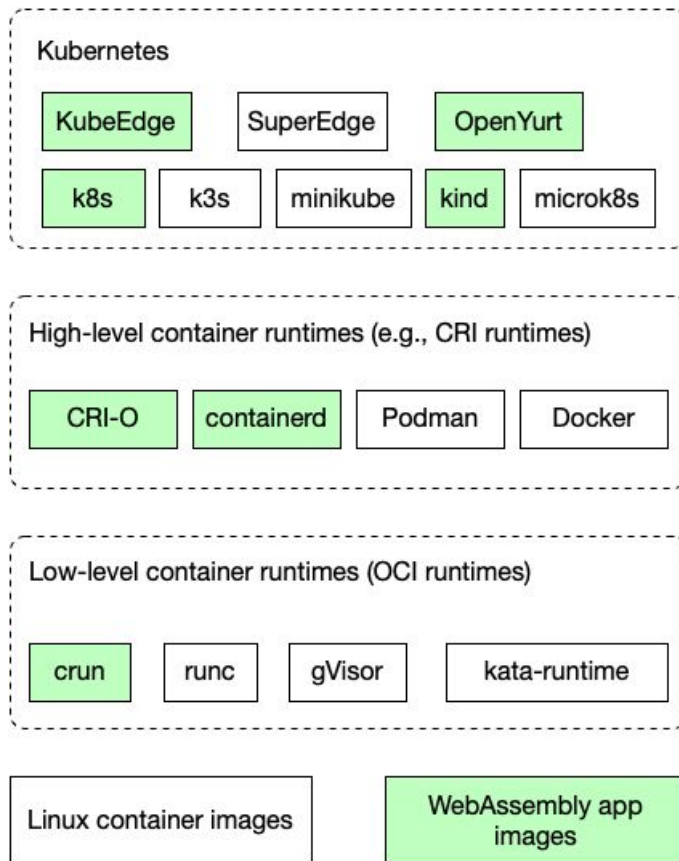
<https://wasmedge.org/book/en/dev/js.html>

- Anna RS
 - KV store optimized for edge use cases
 - Use Rust tokio-based connector to access
- MySQL
 - Use Rust tokio-based clients
 - Use JS clients
- Other cloud databases
 - Many provide MySQL compatible interfaces
 - Or Rust tokio-based client libraries

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

```
git clone https://github.com/containers/crun
cd crun
./autogen.sh
./configure --with-wasmedge
make
sudo make install
```

<https://wasmedge.org/book/en/kubernetes.html>



The container ecosystem

Dapr and service management frameworks

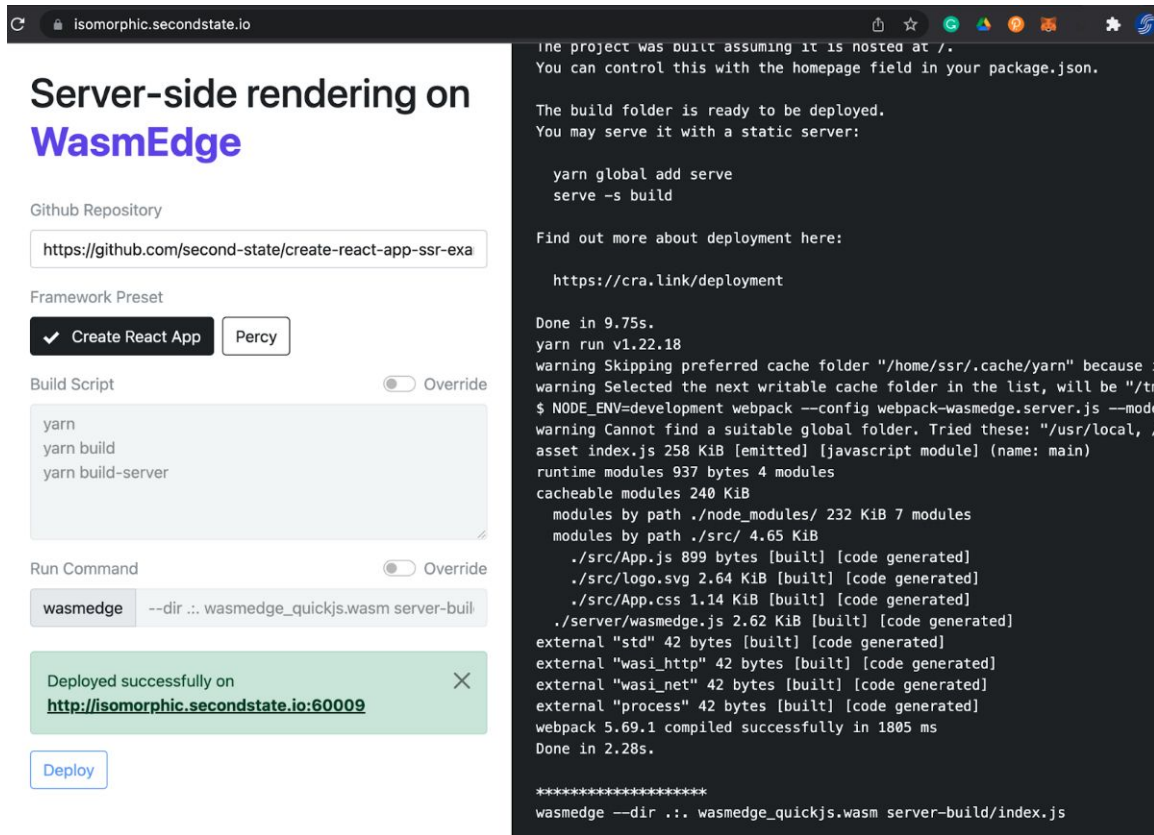
- Dapr
 - Standalone WasmEdge apps as a Dapr sidecar app
 - Communicate with Dapr via sockets using the WasmEdge Dapr SDK
 - <https://github.com/second-state/dapr-wasm>
- essa-rs
 - A stateful FaaS framework based on anna-rs
 - Use Rust SDK to run functions in WasmEdge
 - <https://github.com/essa-project/essa-rs>



Host SDKs

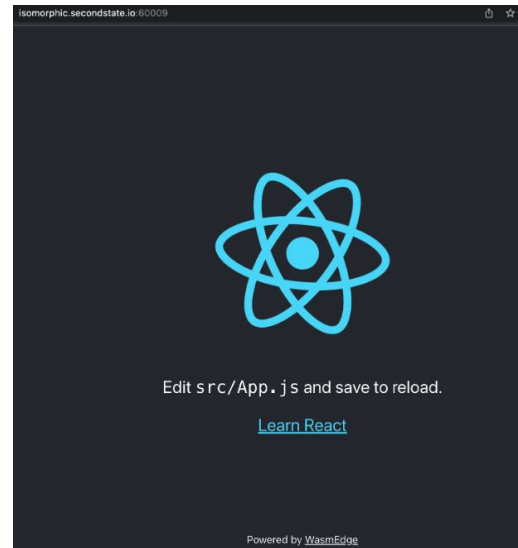
- Rust
- Go
- C
- Python
- Java

Use cases



The screenshot shows the deployment interface for a React application. On the left, there's a form to configure the deployment. The 'Github Repository' field contains `https://github.com/second-state/create-react-app-ssr-exa`. The 'Framework Preset' is set to 'Create React App'. The 'Build Script' section shows `yarn`, `yarn build`, and `yarn build-server`. The 'Run Command' is `wasmedge --dir ../wasmedge_quickjs.wasm server-build`. A green notification box says 'Deployed successfully on <https://isomorphic.secondstate.io:60009>'. A 'Deploy' button is at the bottom.

```
the project was built assuming it is nested 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 it is not writable.  
warning Selected the next writable cache folder in the list, will be "/tmp/.cache/yarn".  
$ NODE_ENV=development webpack --config webpack-wasmedge.server.js --mode=development  
warning Cannot find a suitable global folder. Tried these: "/usr/local, /usr, /usr/share, /usr/share/javascript, /usr/share/nodejs, /usr/share/terminology, /usr/share/terminology/javascript, /usr/share/terminology/nodejs, /usr/share/terminology/nodejs/javascript, /usr/share/terminology/nodejs/javascript/javascript".  
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
```



WasmEdge acts as a lightweight container on the edge cloud for SSR functions in React / Yew etc.

<https://isomorphic.secondstate.io/>



AI inference on the edge

Data streaming framework needs to embed user-defined functions to process streaming camera photos from a factory assembly line to identify defective products.

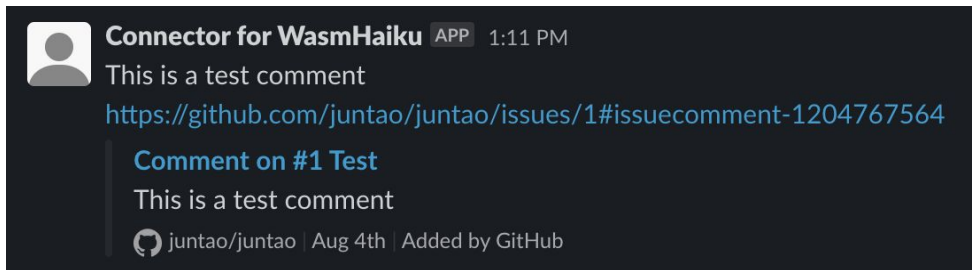
YoMo

Door camera photos need to be processed and identified on a device on or close to the customer's premises for performance and safety reasons.

vmware®

1. Connect an inbound connector (e.g., GitHub)
2. Connect an outbound connector (e.g., Slack)
3. Upload a flow function written in Rust or JavaScript
4. Filter & transform Github notifications before sending to a Slack channel!

<https://docs.wasmhaiku.com/>





HTTP microservice on the edge

```
#[tokio::main(flavor = "current_thread")]
async fn main() -> Result<(), Box<dyn std::error::Error + Send + Sync>> {
    let addr = SocketAddr::from(([0, 0, 0, 0], 3000));

    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);
            }
        });
    }
}

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())),
    }
}
```

Stateful serverless functions on the edge

- Microservices on the backend are also moving to the edge
- General purpose stateful serverless function runtimes
- Technology stack
 - Orchestration: K8s, KubeEdge, OpenYurt, SuperEdge
 - Runtime services: Dapr, Layotto
 - Traffic management: Envoy, MSON, Nginx, APISIX
 - Persistence: AnnaDB, MySQL, and other cloud databases
 - **Runtime container: WasmEdge – standalone runtime for full server apps**
 - Extension services: ESSA, Suborbital, Fermyon

- Serverless functions are the new developer paradigm
 - They are no longer just the "glue" for public cloud services
 - They are increasingly used as microservices to support business logic
 - They should run on edge servers for performance and safety
- Edge cloud serverless functions have unique requirements
 - High performance – low cold start time
 - Small footprint
 - Security and safety
- WasmEdge is the Wasm runtime designed for edge serverless functions



WasmEdgeRuntime

Discuss and learn more:

<https://github.com/WasmEdge/WasmEdge>



CLOUD NATIVE
COMPUTING FOUNDATION



SECOND
STATE

Thanks !