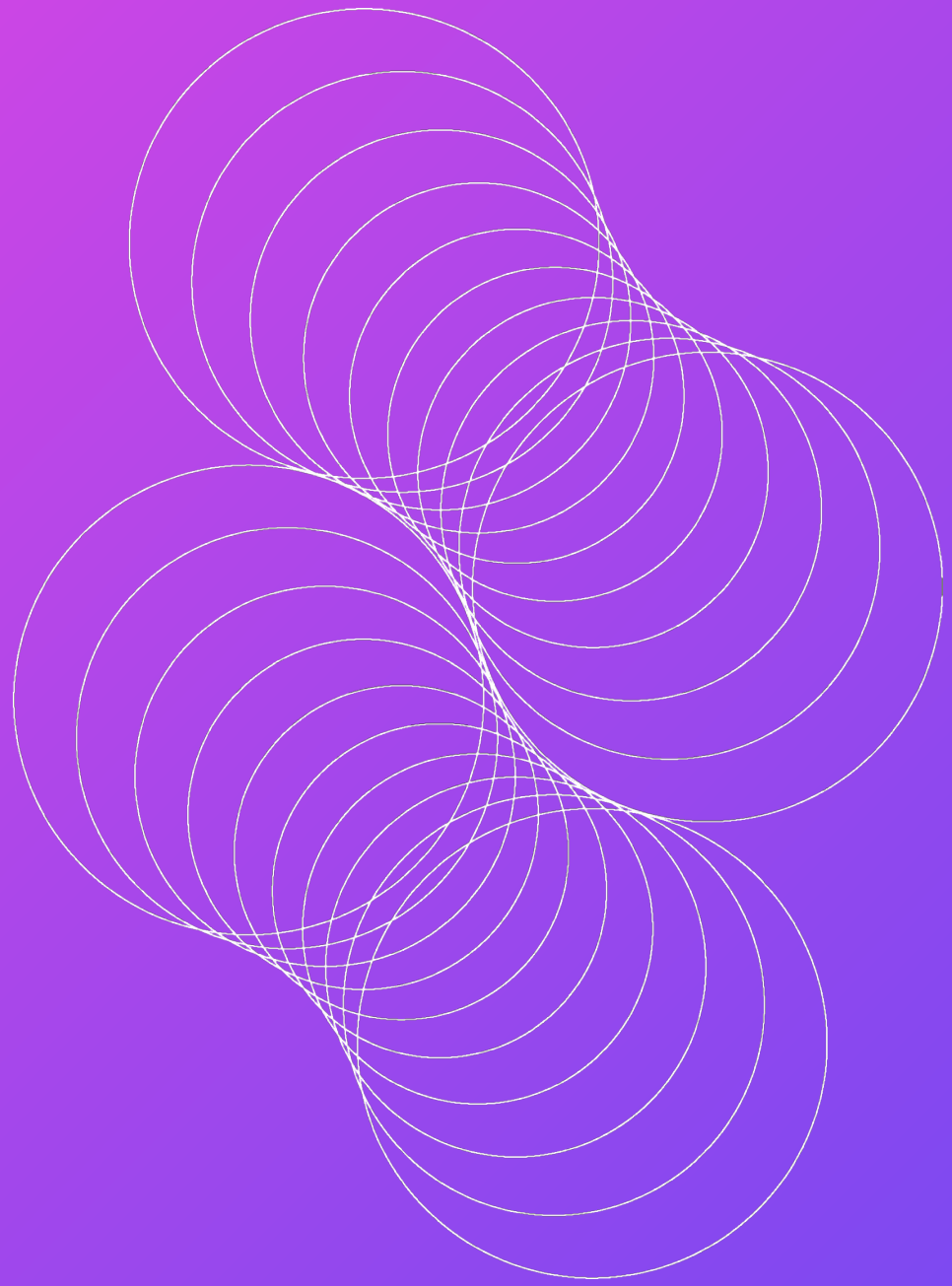
Auth0 by Okta

Go containerless on Kubernetes

With WebAssembly and Rust

Deepu K Sasidharan



Hi, I'm Deepu K Sasidharan

- JHipster co-chair
- Java Champion
- Creator of KDash, JDL Studio, JWT UI
- Developer Advocate @ Okta
- OSS aficionado, polyglot dev, author, speaker

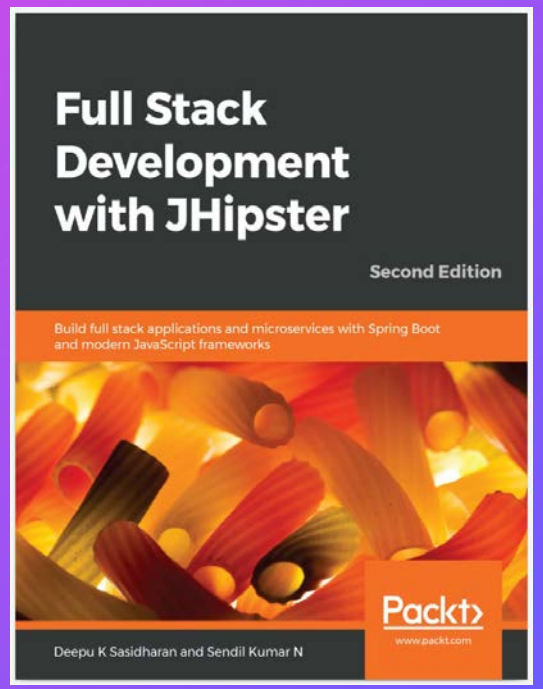
@deepu105@mastodon.social
deepu.tech
@deepu105
deepu05

Auth0 by Okta

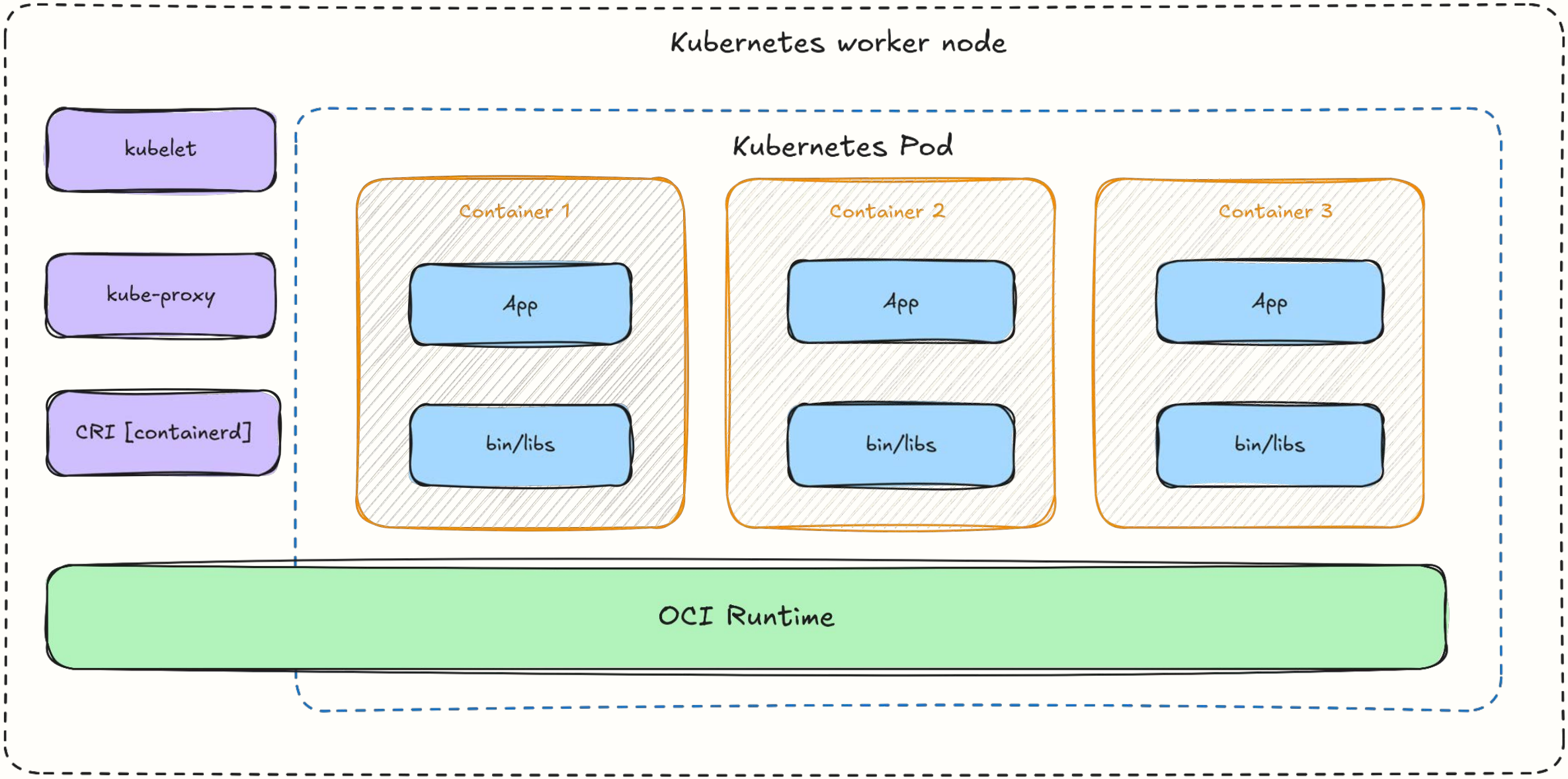
JHipster

TS JS

GO



Containers on Kubernetes


Kubernetes worker node
kubelet
kube-proxy
CRI [containerd]
Kubernetes Pod
Container 1
App
bin/libs
Container 2
App
bin/libs
Container 3
App
bin/libs
OCI Runtime

Advantages of Containers

- Container images are lightweight
- Uses less system resources than a VM
- Portable
- Consistent and reproducible builds
- Faster to boot and run
- Easier to scale and manage
- More secure
- ...

Limitations of Containers

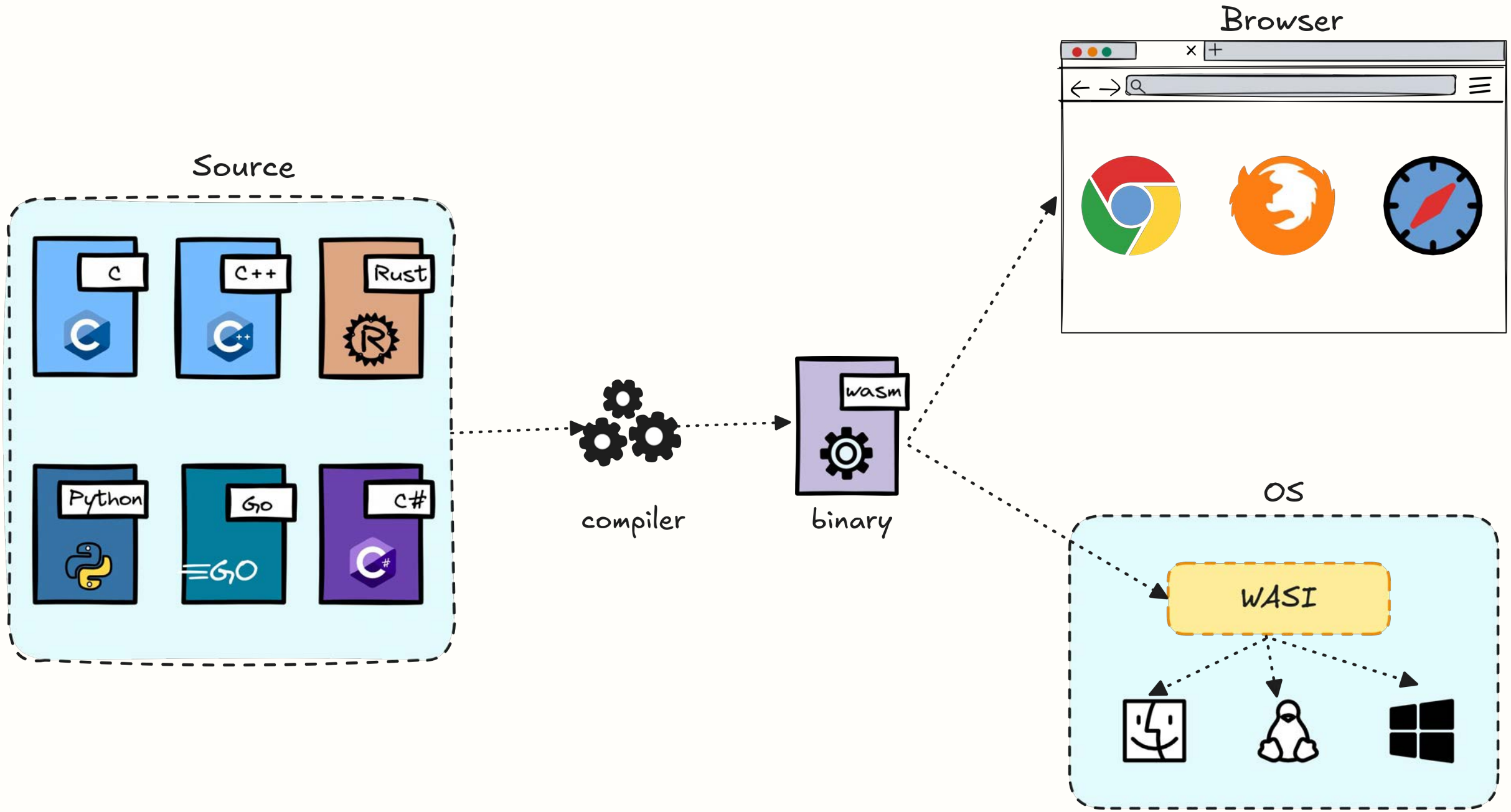
→ Images are OS and architecture dependent

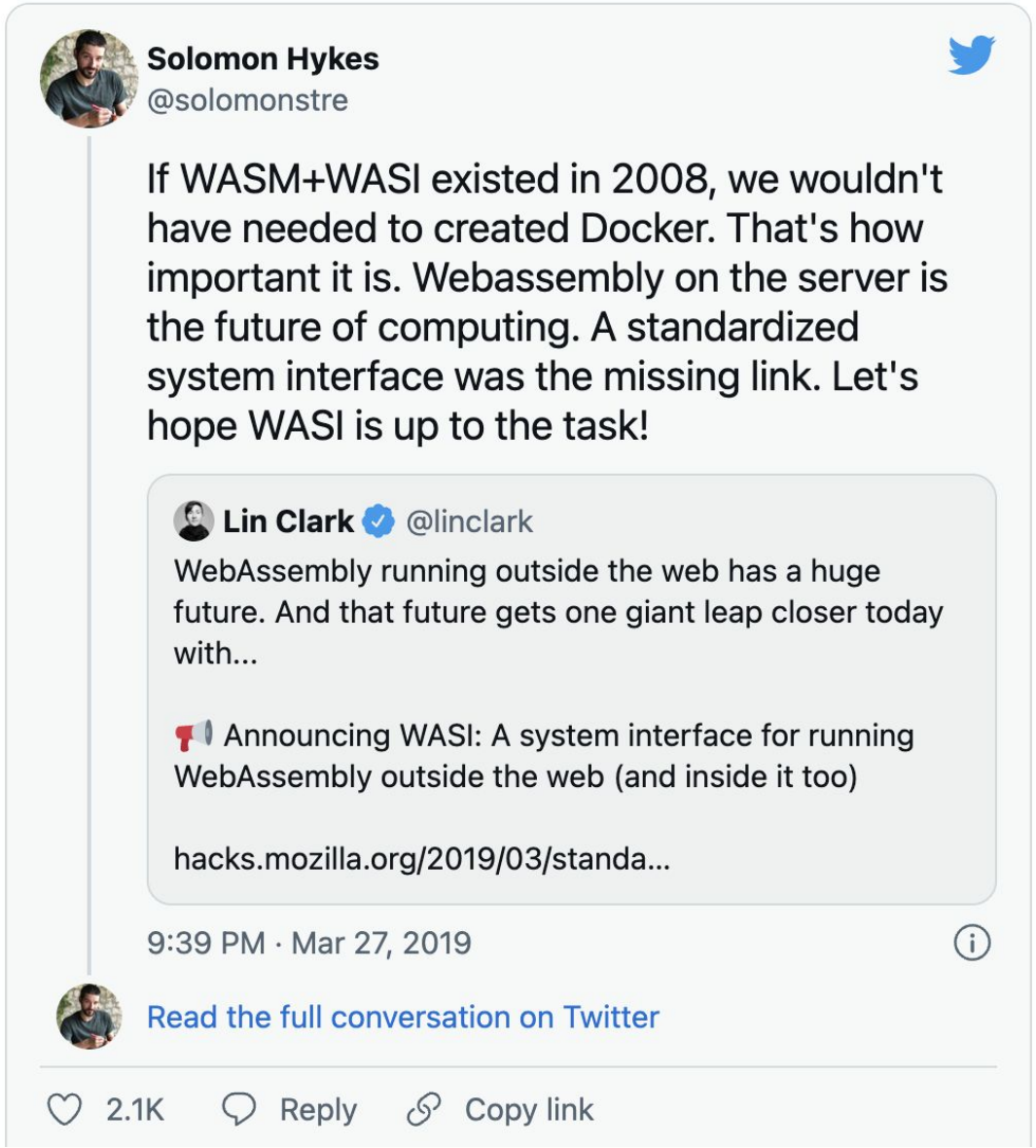
→ Needs a lot of low level OS primitives

→ Does not provide a fine grained control of security sandbox

Containerless == WebAssembly

webassembly.org

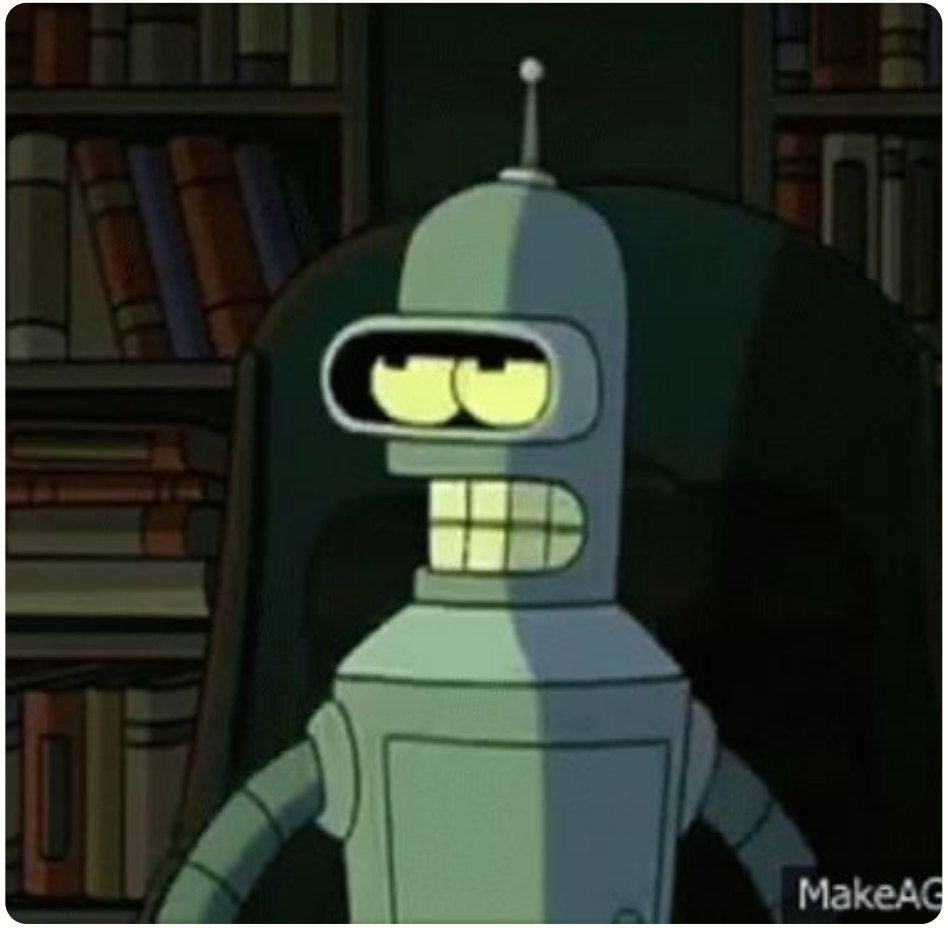

Browser
Source
C
C++
Rust
Python
Go
C#
compiler
wasm
binary
OS
WASI


Solomon Hykes
@solomonstre
If WASM+WASI existed in 2008, we wouldn't have needed to created Docker. That's how important it is. Webassembly on the server is the future of computing. A standardized system interface was the missing link. Let's hope WASI is up to the task!
Lin Clark @linclark
WebAssembly running outside the web has a huge future. And that future gets one giant leap closer today with...
📢 Announcing WASI: A system interface for running WebAssembly outside the web (and inside it too)
hacks.mozilla.org/2019/03/standa...
9:39 PM · Mar 27, 2019
Read the full conversation on Twitter
2.1K
Reply
Copy link

Advantages of Wasm + WASI

- ⟶ Startup speed and near native execution speed
- ⟶ Security isolation (memory isolation and sandbox)
- ⟶ Memory safety and efficiency
- ⟶ Lightweight
- ⟶ Portable(OS/Architecture)

Downsides of Wasm + WASI



→ You can't just take existing projects and make a Wasm binary

→ Resource isolation is still not standardized

→ Networking is still not standardized. No TLS.

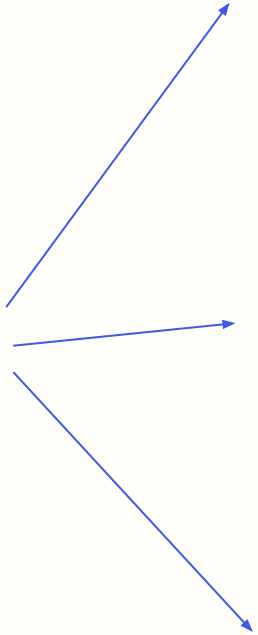
→ Network isolation is still not standardized

→ Multithreading is still in the works

→ Not matured ecosystem

WebAssembly Runtimes

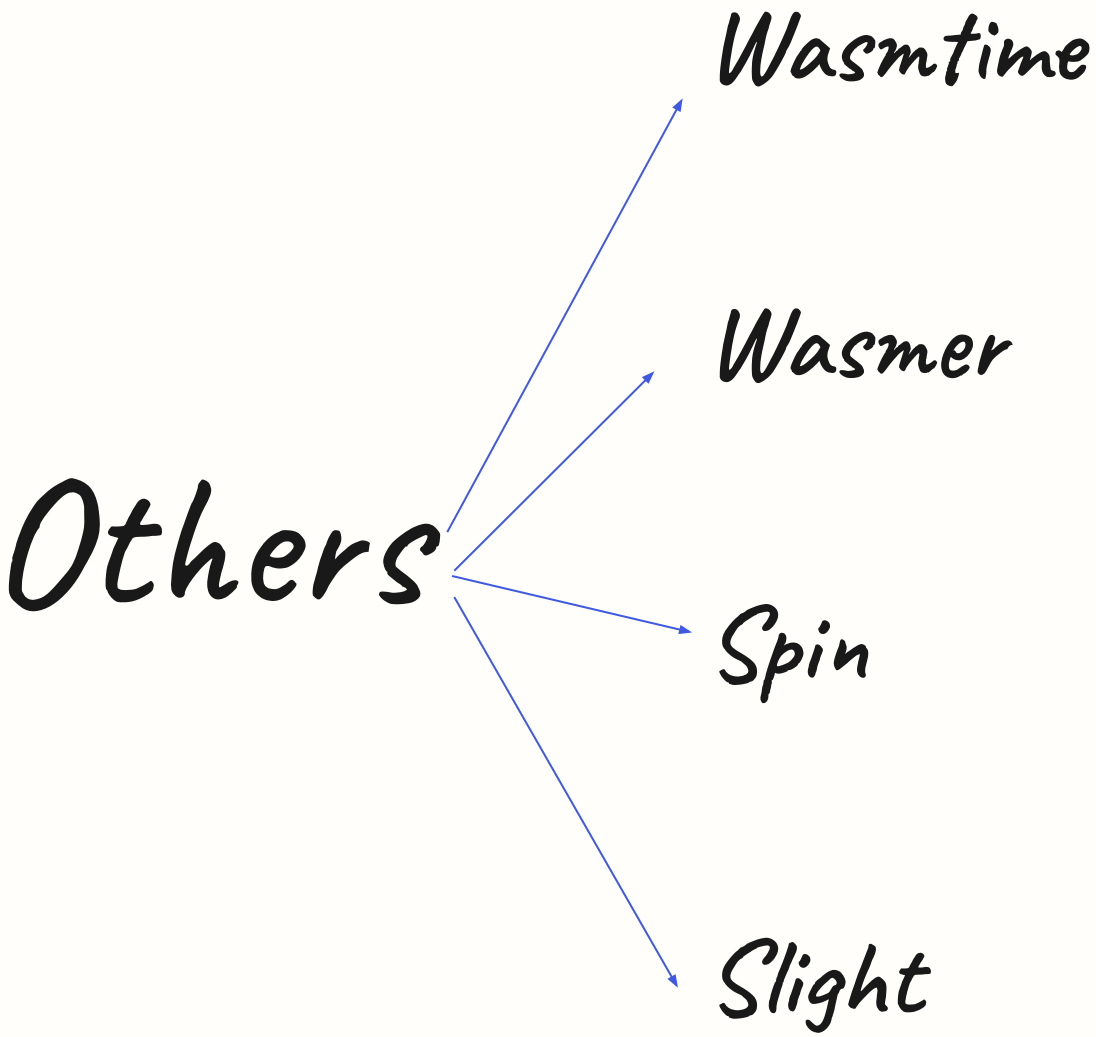
WasmEdge



Widely used in the Kubernetes ecosystem

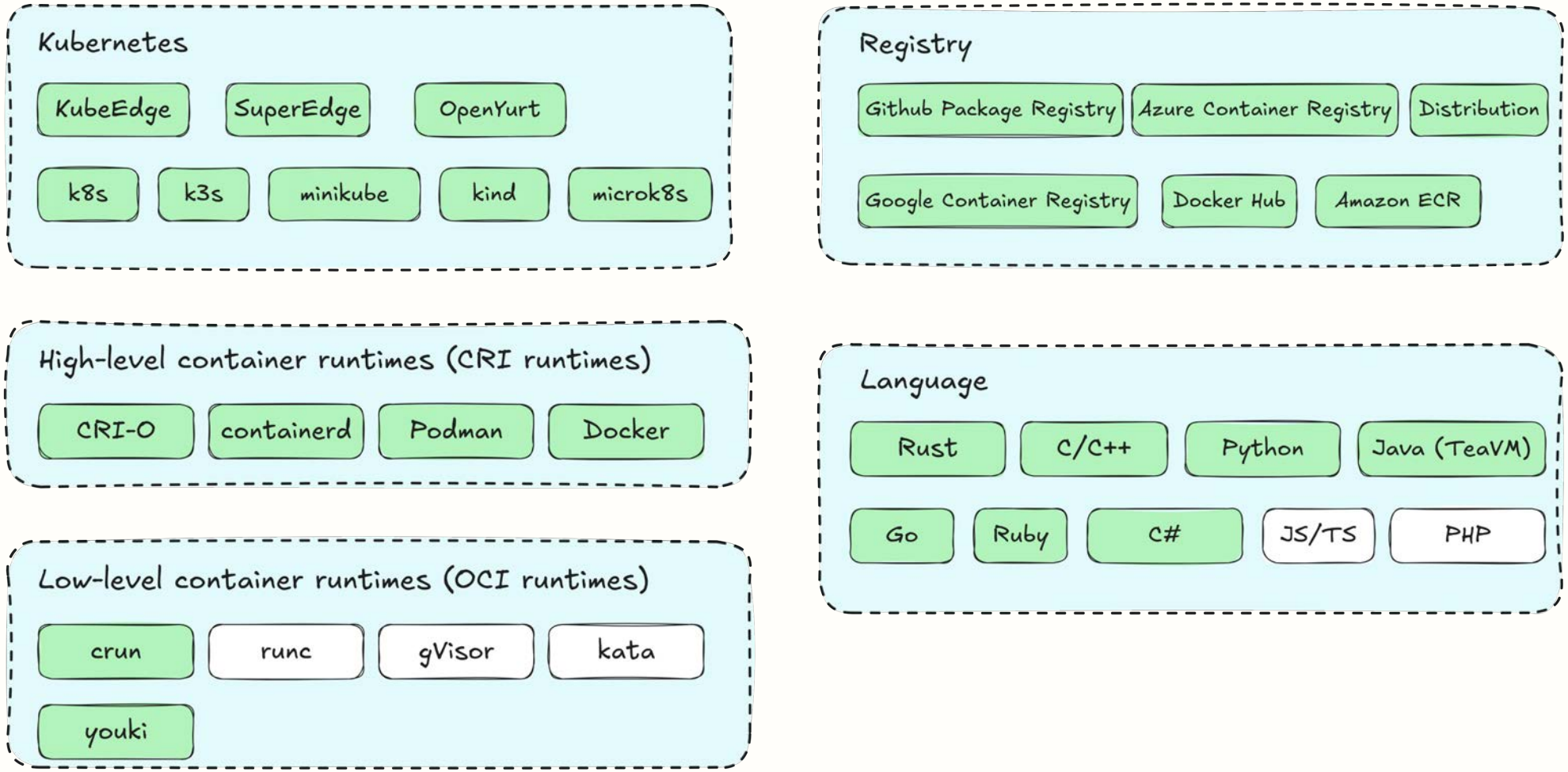
WASI-like extensions

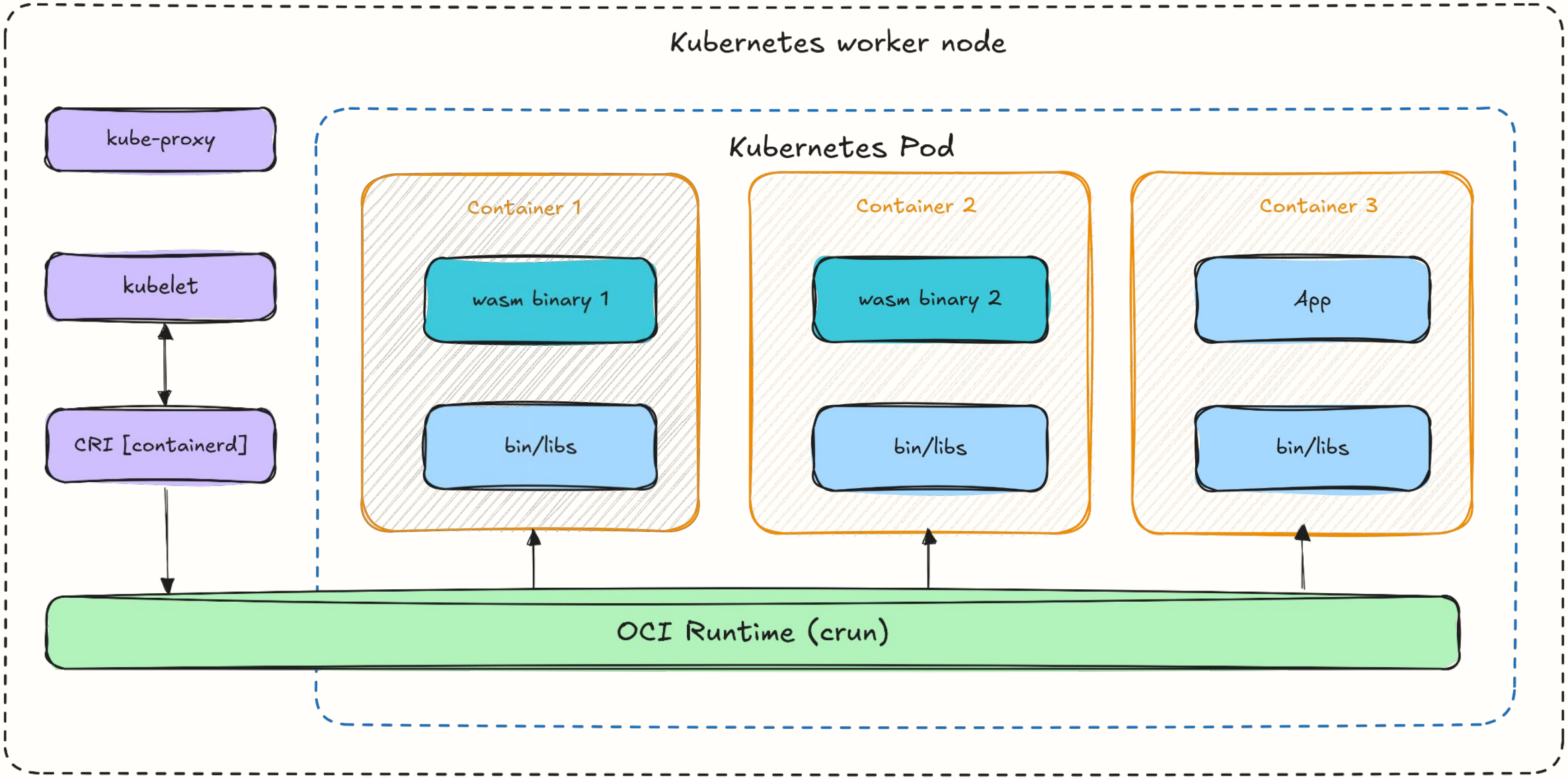
Container tooling, Dapr microservices

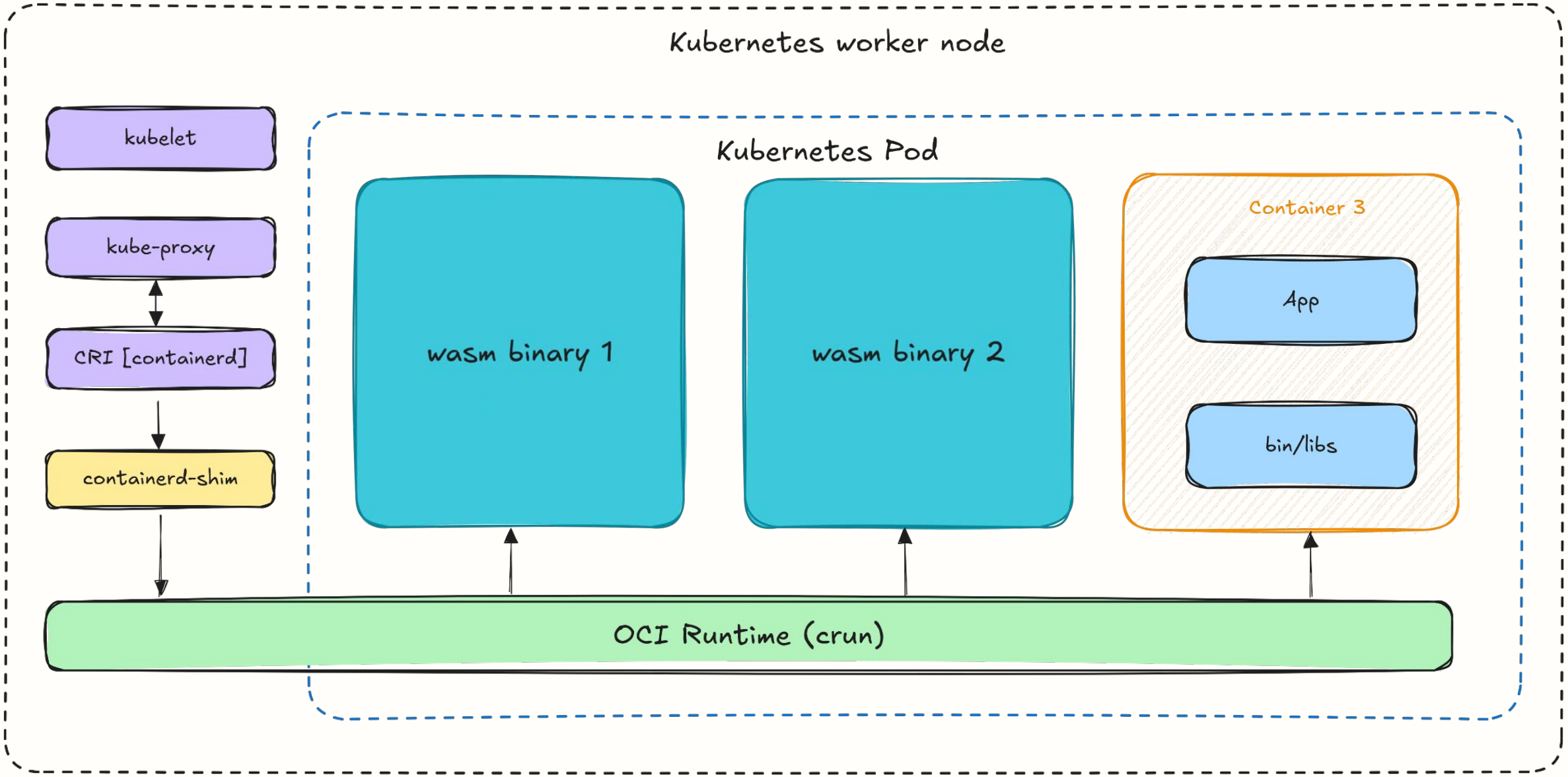

Wasmtime
Wasmer
Others
Spin
Slight

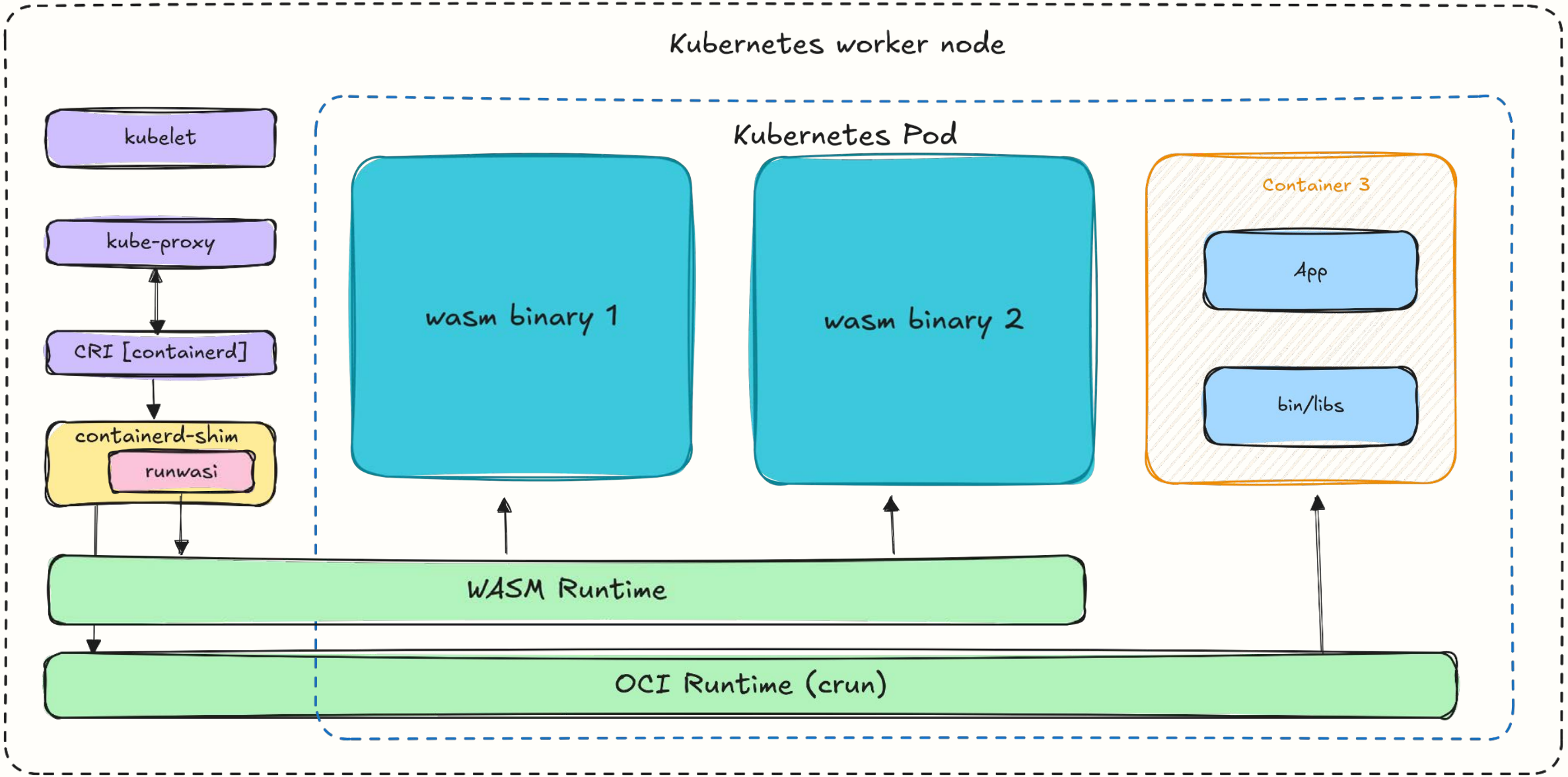
WebAssembly on Kubernetes

Existing Ecosystem Compatibility




Kubernetes worker node
kube-proxy
kubelet
CRI [containerd]
Kubernetes Pod
Container 1
wasm binary 1
bin/libs
Container 2
wasm binary 2
bin/libs
Container 3
App
bin/libs
OCI Runtime (crun)


Kubernetes worker node
kubelet
kube-proxy
CRI [containerd]
containerd-shim
Kubernetes Pod
wasm binary 1
wasm binary 2
Container 3
App
bin/libs
OCI Runtime (crun)


Kubernetes worker node
kubelet
kube-proxy
CRI [containerd]
containerd-shim
runwasi
Kubernetes Pod
wasm binary 1
wasm binary 2
Container 3
App
bin/libs
WASM Runtime
OCI Runtime (crun)

Rust + WebAssembly

- → Performance: Low-Level Control with High-Level Ergonomics
- → First class WebAssembly support
- → Smaller binaries
- → Great ecosystem for WebAssembly
- → Great Developer Experience

Demo time

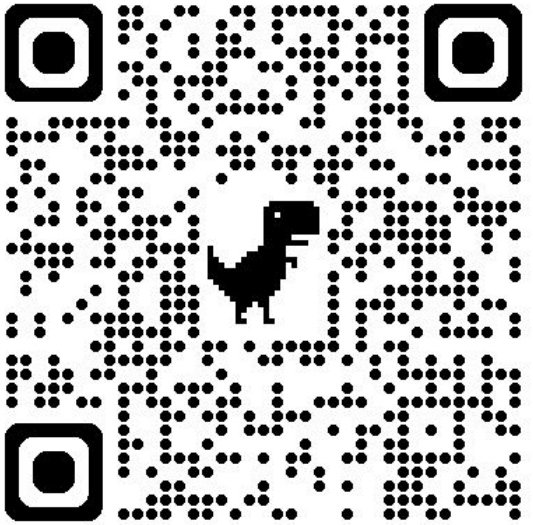
```
# Create a Rust WASM app
$ clone https://github.com/deepu105/word-generator-wasi-rust.git

# Build for WASM
$ rustup target add wasm32-wasi
$ cargo build --target wasm32-wasi --release

# Install WasmEdge for testing →
https://wasmedge.org/docs/start/install

# Run the server locally
$ wasmedge target/wasm32-wasi/release/weather_forecast.wasm

# Open another terminal and test the server
$ curl "http://localhost:8090?starts_with=d"
```



https://github.com/deepu105/word-generator-wasi-rust

```
# Install buildah for creating OCI images →
https://github.com/containers/buildah/blob/main/install.md

# Build an OCI Image
$ buildah build --annotation
"module.wasm.image/variant=compat-smart" -t
word_generator_wasi Dockerfile_OCI
# Push the OCI Image to Docker Hub
$ buildah push --authfile ~/.docker/config.json
word_generator_wasi
docker://docker.io/deepu105/word_generator_wasi:latest

# Create a "KinD" Cluster
$ kind create cluster
# Enable WasmEdge support using KWasm →
https://kwasm.sh/quickstart/

# Deploy the app
$ kubectl apply -f k8s-manifest.yaml
```

https://github.com/deepu105/word-generator-wasi-rust

Future?

→ WebAssembly on Kubernetes and cloud is still evolving

→ Rust is an ideal language for Wasm+WASI

→ Many features are being standardized

→ Machine learning support is coming

→ Many new features

→ Containerless is definitely on the horizon

okta

The World's Identity Company

dev_day

a 24 hour virtual event

September 24, 2024

Registration opens summer of 2024
Stay up to date with the latest at: a0.to/devday

Thank You

SCAN ME

Subscribe to our newsletter

a0.to/nl-signup/java