CLOUD NATIVE
Wasm DAY
NORTH AMERICA

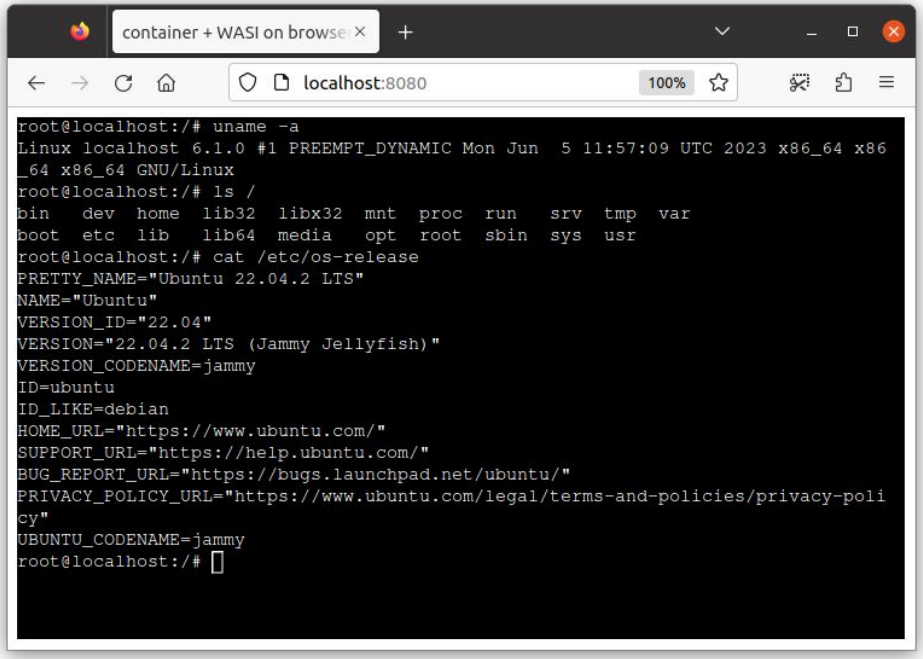
Running Linux-Based Containers on Wasm and Browser with Container2wasm Converter

Kohei Tokunaga, NTT

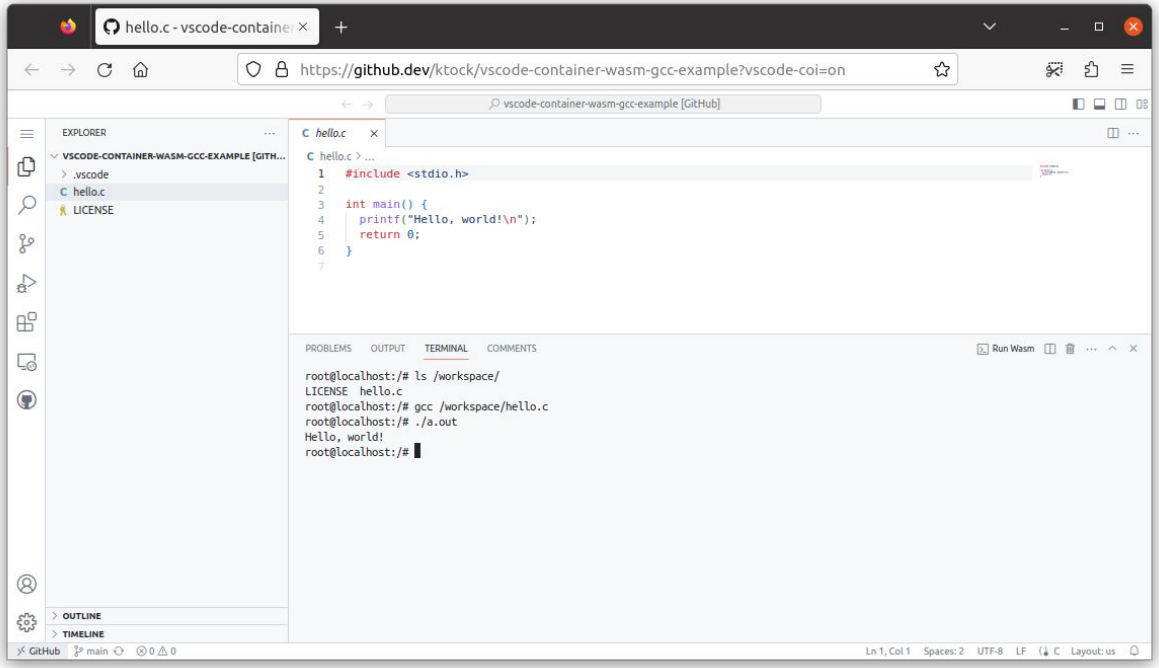
Summary

- Porting apps to Wasm costs time for re-compilation and/or re-implementation
- Container2wasm enables running unmodified containers on Wasm, leveraging CPU emulators
- Created an extension of VSCode for the Web to run containers on browser

Debian container on browser



Debian container on `github.dev`



Motivations to port apps to Wasm

- Leveraging existing apps on browser (e.g. for dev environment, as a building block, for demo page, …)
  - Ruby.wasm
  - VSCode Python for the web
  - WordPress on browser
  - Sqlite3 on browser
  - postgres-wasm

- Leveraging Wasm features for existing applications
  - Sandboxed and portable execution environment
  - Pre-initialization by Wizer
  - Record & Replay by Timecraft

But, porting apps to Wasm is hard

CLOUD NATIVE
Wasm DAY
NORTH AMERICA

- Requires re-compilation and/or re-implementation
  - Binary format difference
  - Need (re)designing of apps for wasm architecture
  - Kernel (e.g. Linux) features/APIs not fully available on Wasm(Wasi)

- Existing approaches rely on compiler's Wasm target (emscripten, wasi-sdk, Go, …)
  - But lacks common features of existing system (e.g. no fork/exec)

- Can we run unmodified applications on Wasm?

container2wasm converter

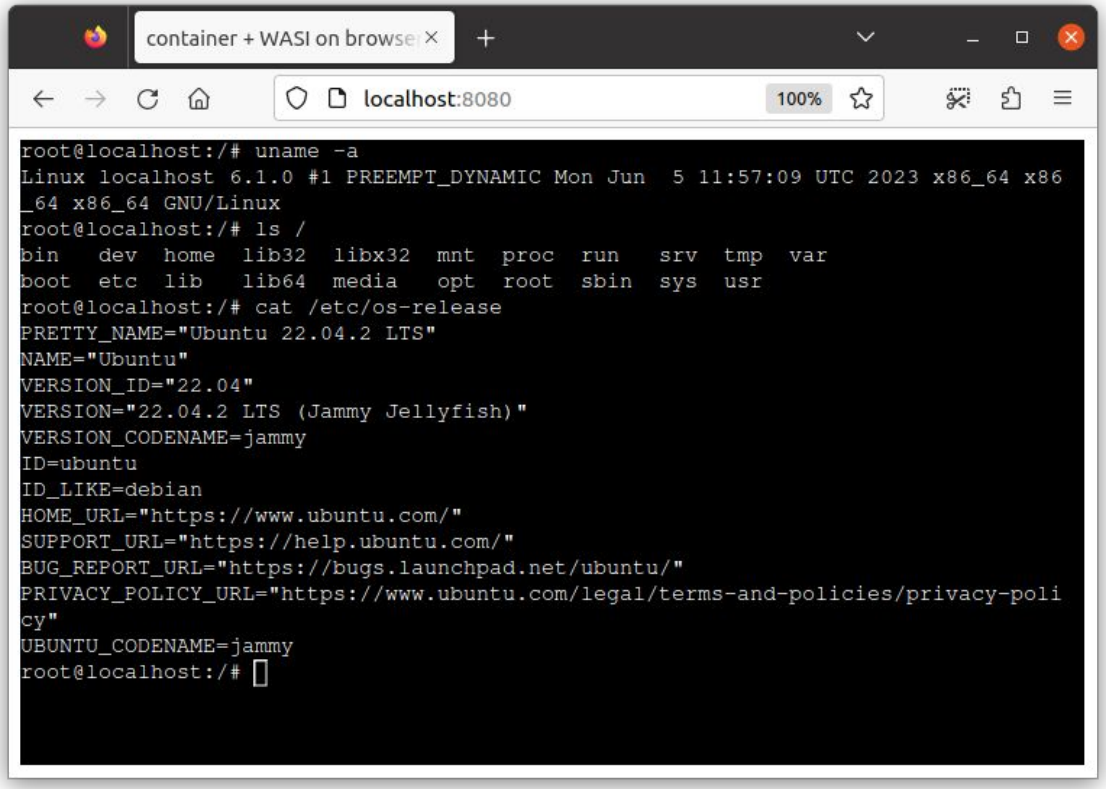

```
$ c2w ubuntu:22.04 ubuntu.wasm
```

- An experimental **Container** to **Wasm** converter
- Runs unmodified Linux-based containers on Wasm
  - Containers run on real Linux on emulated processor (x86_64 or RISC-V)
- Containers run on Wasi runtimes and on browser

Container on Wasi runtime

```
$ wasmtime ubuntu.wasm uname -sm
Linux x86_64
```

Container on browser



Containers on Wasi runtimes

- Outputs Wasi image by default
- Tested on popular runtimes
  - wasmtime, wamr, wasmer, wasmedge, wazero
- Features
  - Stdio and envvar
  - Directory mapping
  - Networking (w/ NW stack outside of the runtime)
  - etc…

runtime	stdio	env	mapdir	NW
wasmtime	✔	✔	✔	✔*
wamr	✔	✔	✔	🚧
wazero	✔	✔	✔	✔*
wasmer	🚧 (stdin WIP)	✔	✔	🚧
wasmedge	🚧 (stdin WIP)	✔	✔	🚧

* relies on NW stack outside of the runtime

```
$ c2w ubuntu:22.04 out.wasm
$ wasmtime out.wasm
root@localhost:/# uname -srm
uname -srm
Linux 6.1.0 x86_64
root@localhost:/# ls /
ls /
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr
```

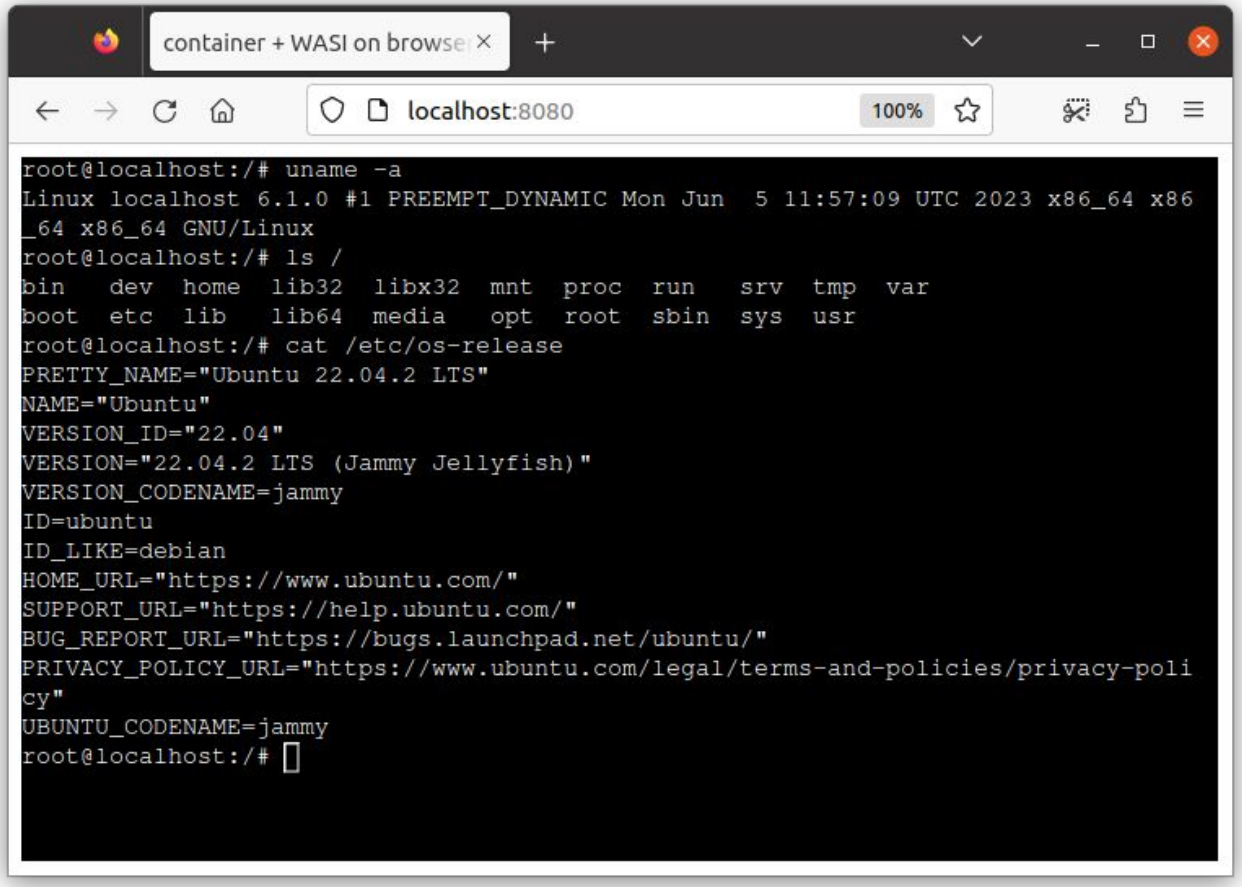
```
$ wasmtime --mapdir /mnt/share::/tmp/share out.wasm cat /mnt/share/from-host
hi
```

Containers on browser

Two configurations are available:

- **Running Wasi image on browser**
  - w/ several existing host implementations
    - bjorn3/browser_wasi_shim
    - microsoft/vscode-wasm
    - …
- **Emscripten**
  - container startup can be slow (about 30s) because pre-initialization isn't enabled

Networking is also available (discussed later)



Networking (Wasi runtimes)

- Containers converted to Wasi image can perform networking
- Relies on networking stack `c2w-net` running outside of Wasi runtime
- Container can access to anywhere accessible from the network stack

```
$ c2w-net --invoke /tmp/out/out.wasm --net=socket sh
connecting to NW...
INFO[0001] new connection from 127.0.0.1:1234 to 127.0.0.1:50470
/ # apk update && apk add --no-progress figlet
apk update && apk add --no-progress figlet
apk update && apk add --no-progress figlet
fetch https://dl-cdn.alpinelinux.org/alpine/v3.18/main/x86_64/APKINDEX.tar.gz
fetch
https://dl-cdn.alpinelinux.org/alpine/v3.18/community/x86_64/APKINDEX.tar.gz
v3.18.3-149-g8225da85c11 [https://dl-cdn.alpinelinux.org/alpine/v3.18/main]
v3.18.3-151-g6953e6f988a [https://dl-cdn.alpinelinux.org/alpine/v3.18/community]
OK: 20071 distinct packages available
(1/1) Installing figlet (2.2.5-r3)
Executing busybox-1.36.0-r9.trigger
OK: 8 MiB in 16 packages
/ # figlet hello
figlet hello
figlet hello
 _          _ _
| |__   ___| | | ___
| '_ \ / _ \ | |/ _ \
| | | |  __/ | | (_) |
|_| |_|\___|_|_|\___/
```

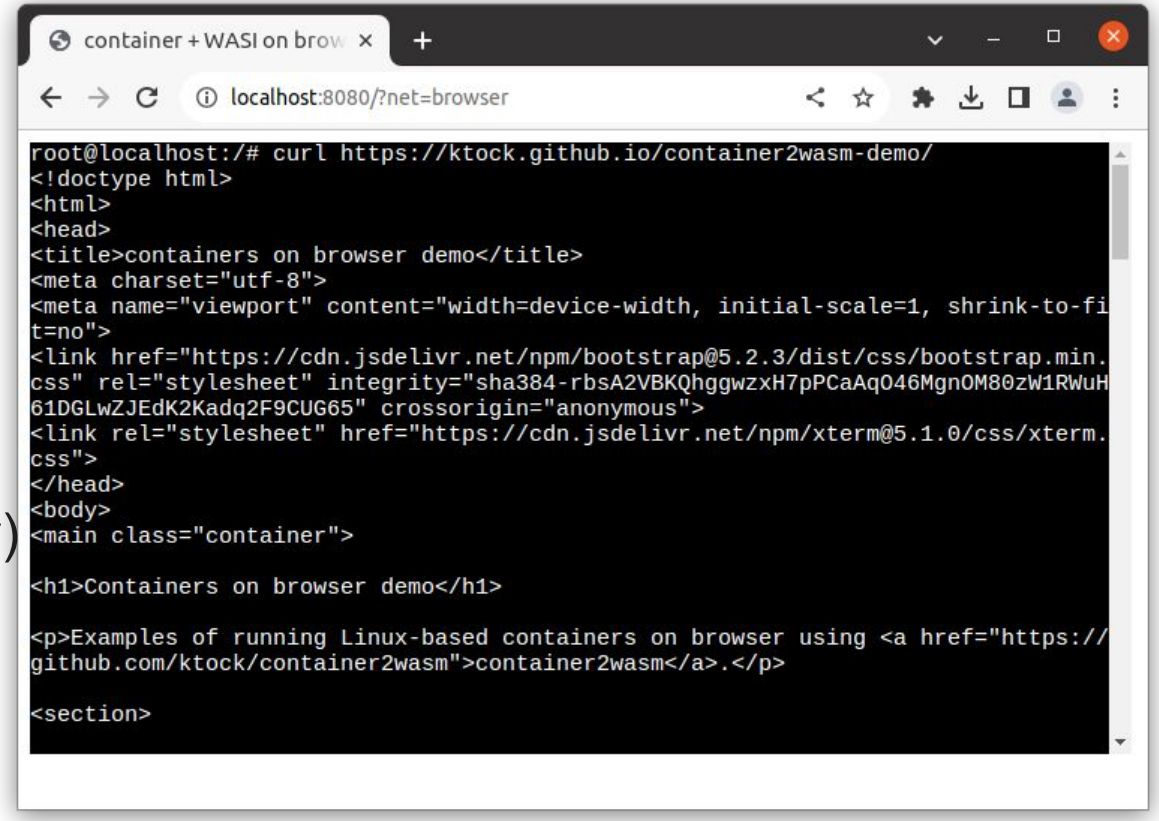
Supports wasmtime and wazero as of now

Networking (on browser)

Two configurations are available:

- **Running NW stack outside of browser**
  - Pros: Accessible sites not limited by browser
  - Cons: Maintenance cost of NW stack on the host

- **Running NW stack on browser**
  - Pros: Easy to maintain (no host-side dependency)
  - Cons: HTTP(S) only. Restrictions by browser (CORS, Forbidden Headers)



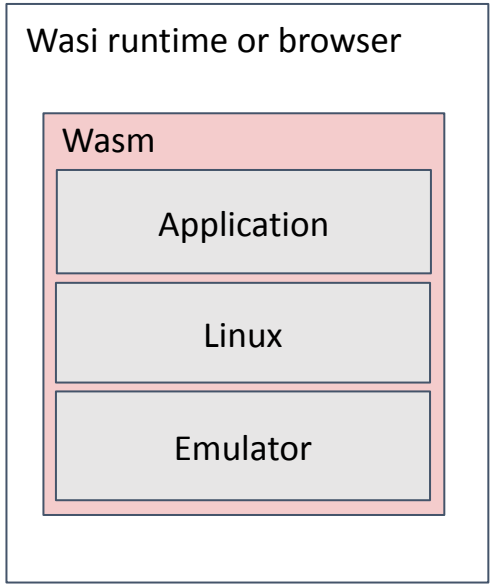
Demo: Container on browser

CLOUD NATIVE
Wasm DAY
NORTH AMERICA

- Demo Page: https://ktock.github.io/container2wasm-demo/
- container2wasm Repo: https://github.com/ktock/container2wasm

How it works

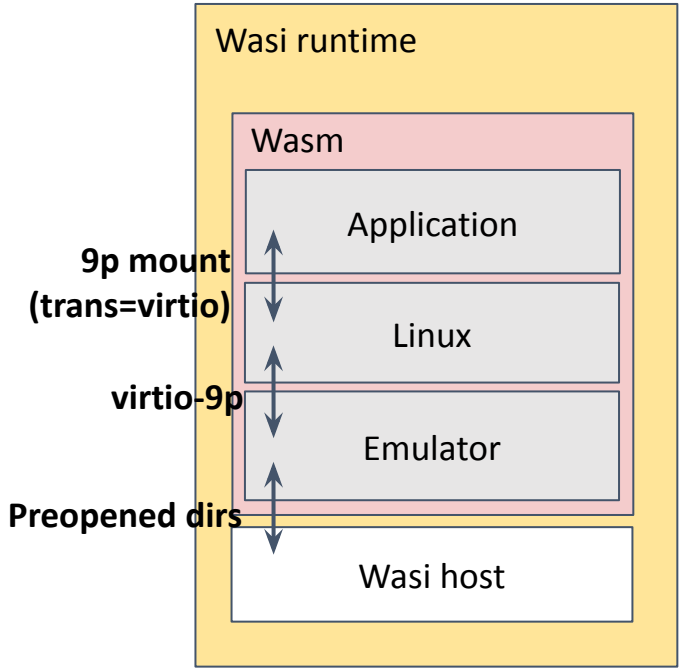
- Conversion steps written in Dockerfile (embedded to `c2w` command)
- Container and Linux kernel run on CPU emulator compiled to Wasm
  - Bochs (for x86_64 containers)
  - TinyEMU (for RISC-V containers)
- Dependencies (emulator, kernel, container rootfs, runc, etc.) are packaged into a single WASM image
  - WASI: wasi-vfs is used
  - emscripten : `--preload-file` is used
- Kernel is pre-booted during build time by Wizer (experimental, WASI only)



How it works: Directory mapping (Wasi)

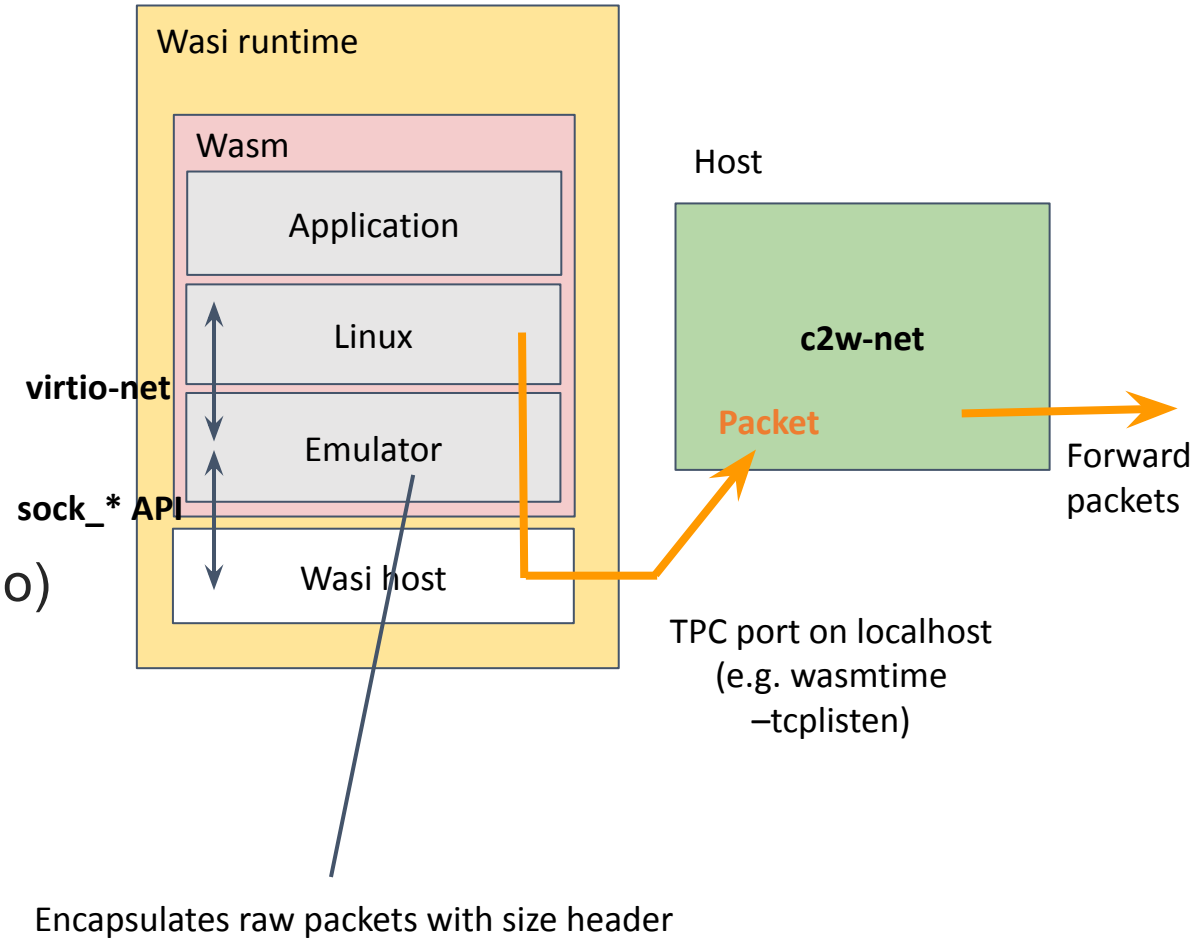
CLOUD NATIVE Wasm DAY NORTH AMERICA

- Emulator sees pre-opened (mapped) dirs via `fd_*` APIs of Wasi
- They are shared to the guest OS via virtio-9p



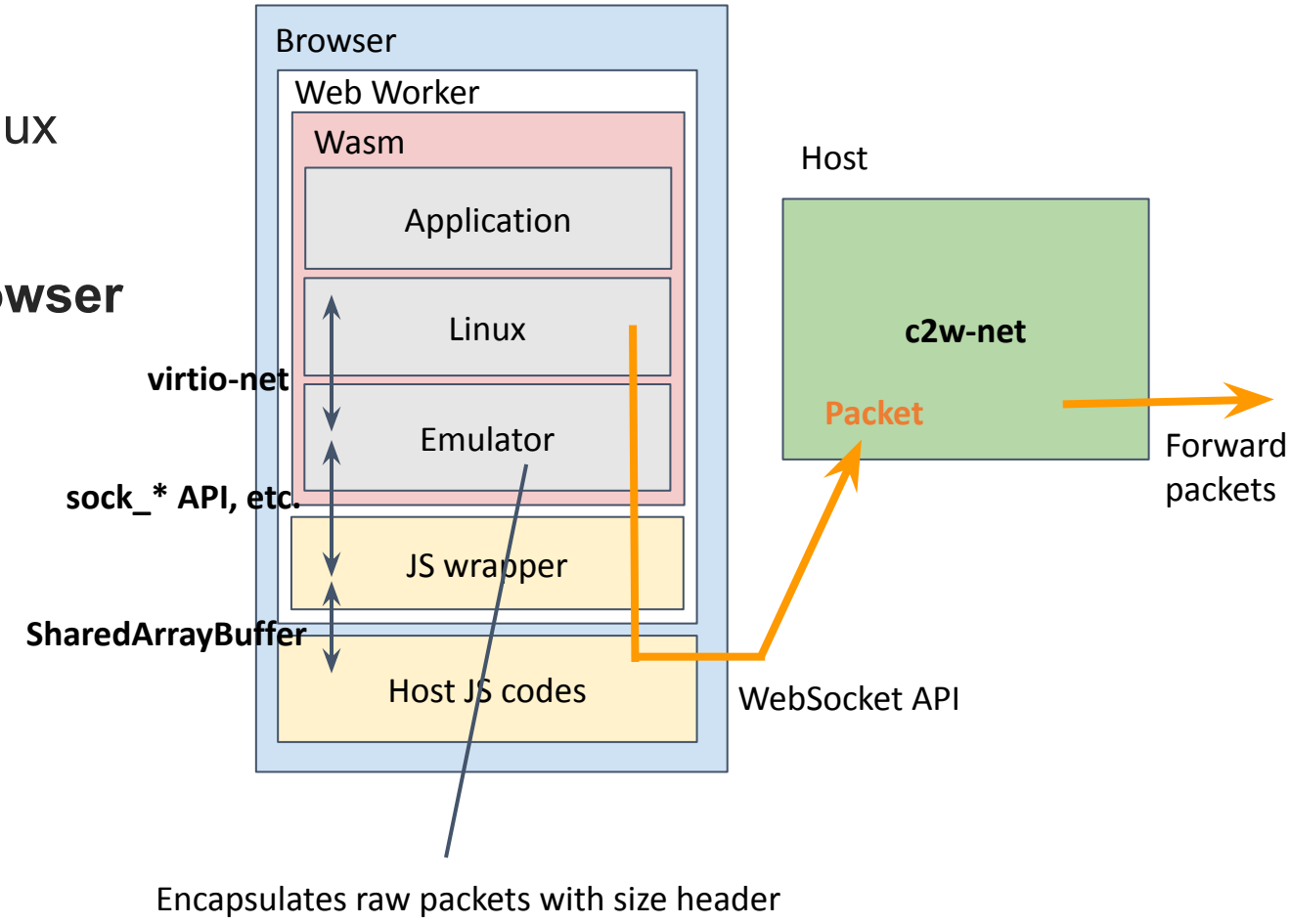
How it works: Networking (Wasi)

- Virtio-net device provided to the guest Linux
- Emulator forwards packets relying on `c2w-net`
  - NW stack running **outside of the runtime**
  - Connected over Wasi's `sock_*` APIs
- `c2w-net` uses gvisor-tap-vsock (NW stack in Go)



How it works: Networking (On browser + WebSocket)

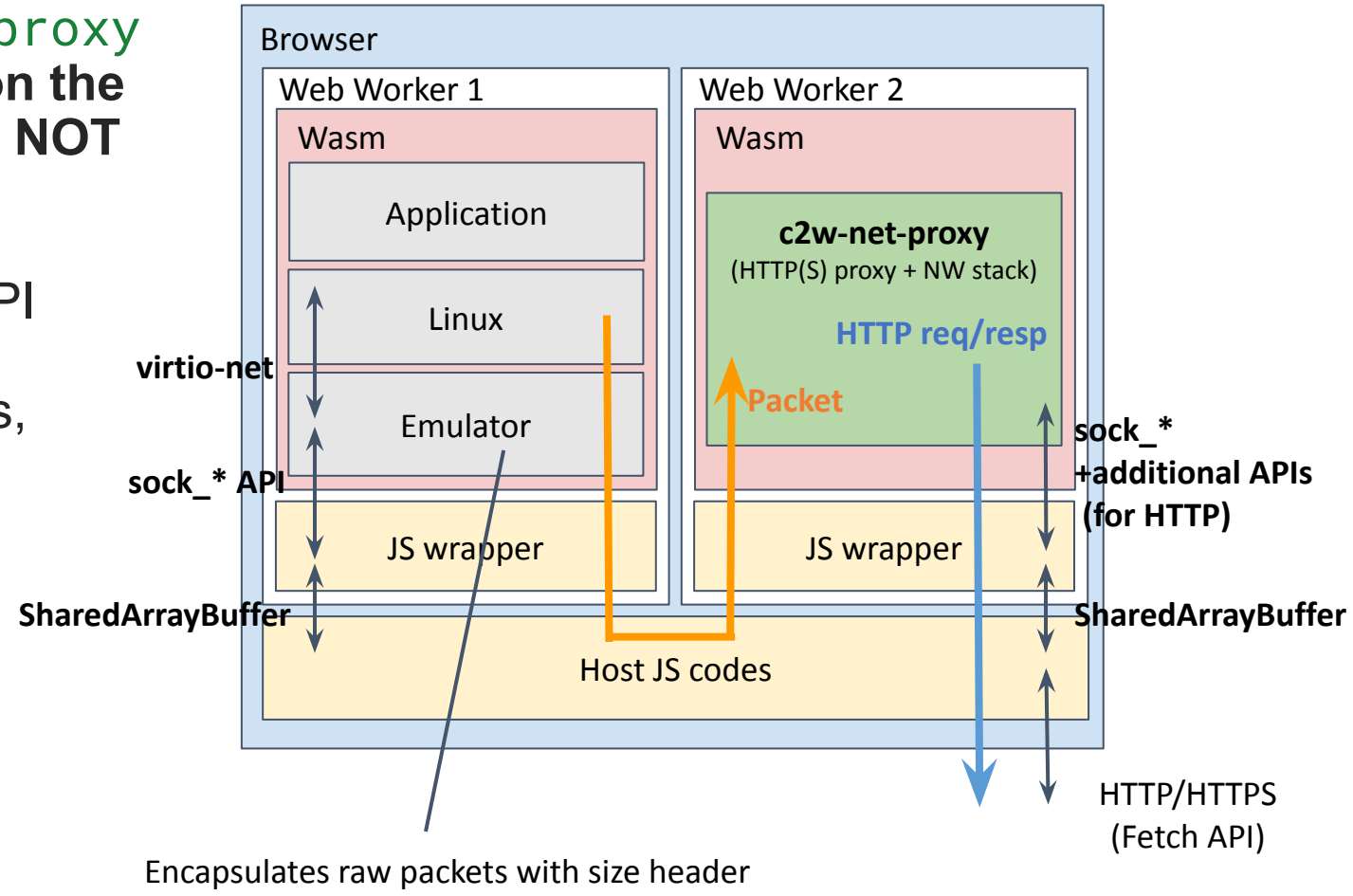
- Virtio-net device provided to the guest Linux
- Emulator forwards packets via `c2w-net`
  - NW stack running **outside of the browser**
  - Connected over WebSocket API



How it works: Networking (On browser + Fetch API)

- Emulator forwards packets via `c2w-net-proxy`
  - NW stack + HTTP(S) proxy **running on the browser (host-side NW forwarder is NOT needed)**
- Forwards HTTP(S) packets using Fetch API
- Terminates HTTPS with its own certificates, re-encrypting by Fetch API
- Restrictions by Fetch API
  - Accessible sites limited by CORS
  - Forbidden Headers can't be controlled

Supports only
WASI-on-browser as of now



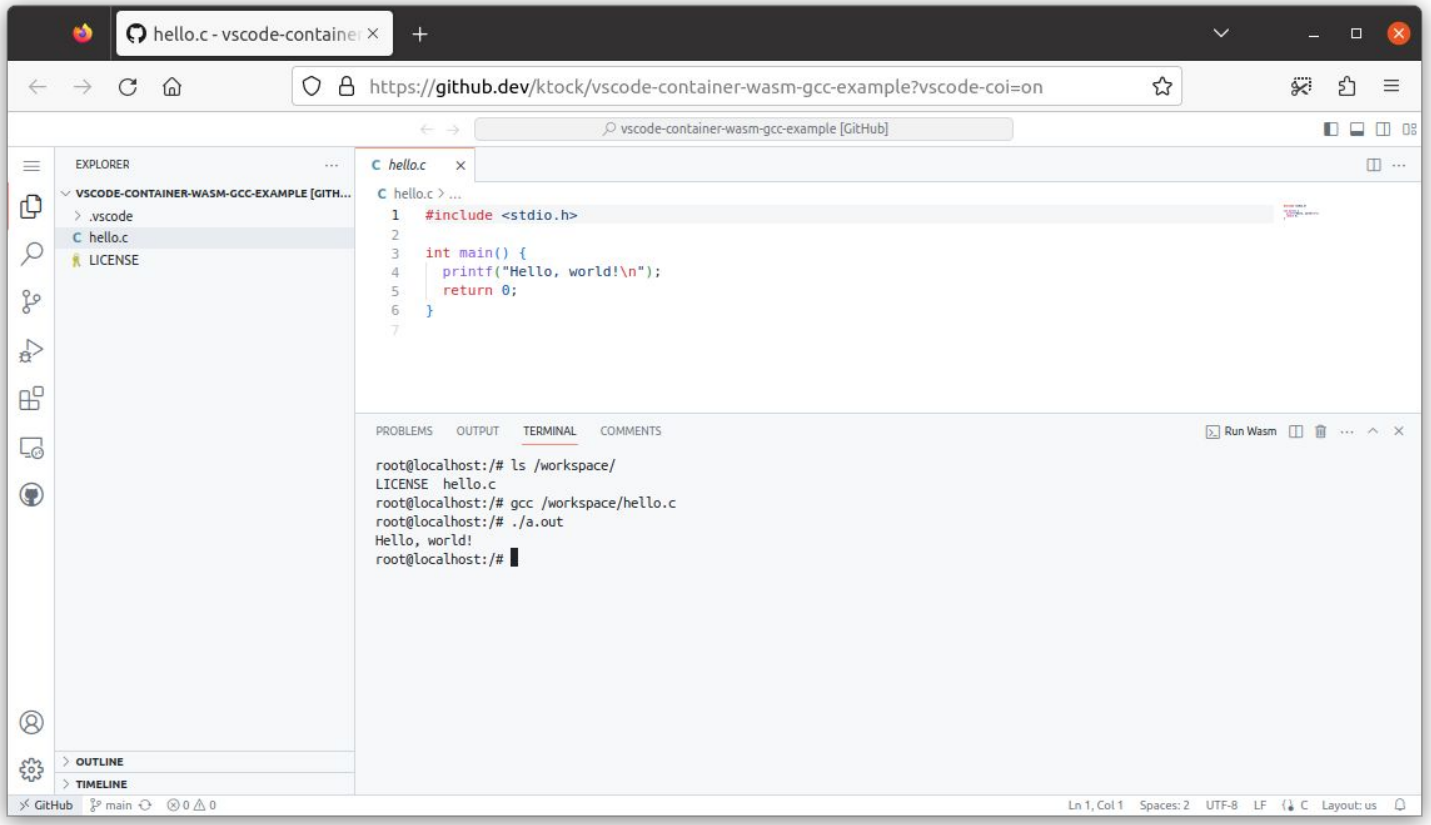
Running containers on VSCode for the Web

CLOUD NATIVE Wasm DAY NORTH AMERICA

https://github.com/ktock/vscode-container-wasm

- An extension of VSCode for the Web to run containers on browser
  - `ktock.container-wasm`
- Containers run **on browser** so no need to prepare remote containers
- Workspace is mounted at `/workspace/`
- Networking available based on Fetch API (w/ restrictions by browser like CORS)
- Uses microsoft/vscode-wasm for Wasi host for containers

Debian container on `github.dev`



https://github.com/ktock/vscode-container-wasm-debian-example

Demo of containers on VSCode for the Web

CLOUD NATIVE
Wasm DAY
NORTH AMERICA

- Demo Page: https://github.com/ktock/vscode-container-wasm-gcc-example
- vscode-container-wasm Repo: https://github.com/ktock/vscode-container-wasm

Other possible use cases

CLOUD NATIVE
Wasm DAY
NORTH AMERICA

- Interactive on-browser linux-based demo
- On-browser IDEs
- Sandboxed execution environment of containers
- Application debugger runnable on browser
- Record & Replay debugging
- etc…

Future works

- Performance analysis & improvement
  - Emulation performance
  - Conversion speed
  - Image loading speed
  - Networking performance
  - Image size
- Accessing to OS package repos (e.g. apk, apt, fedora packages, …) from browser
  - Repos need to allow CORS access
- Usability improvement
  - Automatic conversion on OCI registries
  - Emitting Wasm image from Dockerfile builders
  - NW stack purely on Wasi runtime
- Graphics support

Related works

VMs on browser

- **v86**: https://github.com/copy/v86
  - x86-compatible CPU emulator running on browser created by Fabian Hemmer
  - Supports wide variety of guest OSes (including Windows)
  - No support for x86_64, not target to Wasi
- **TinyEMU**: https://bellard.org/tinyemu/
  - RISC-V and x86 emulator created by Fabrice Bellard
  - Can run on browser
  - Container2wasm uses this for RISC-V emulation
  - No support for x86_64, not target to Wasi

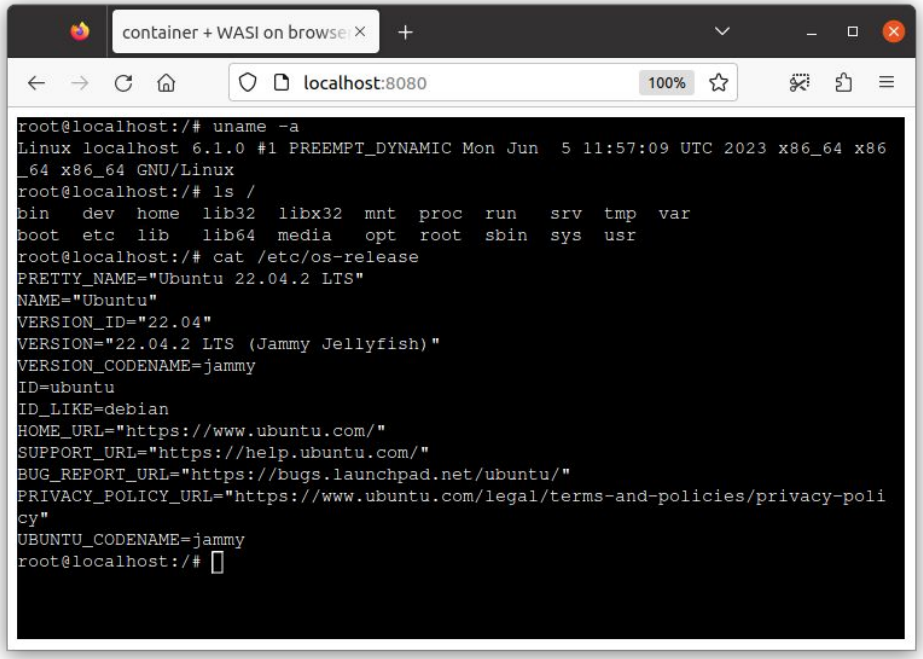
Binary translation

- **MyAOT (Name is subject to change)**: https://github.com/AkihiroSuda/myaot
  - Translator of a Linux/riscv ELF binary to Wasm, proposed by Akihiro Suda, NTT
  - No CPU emulator
  - Experimental status (only trivial programs work, syscalls are not fully implemented)

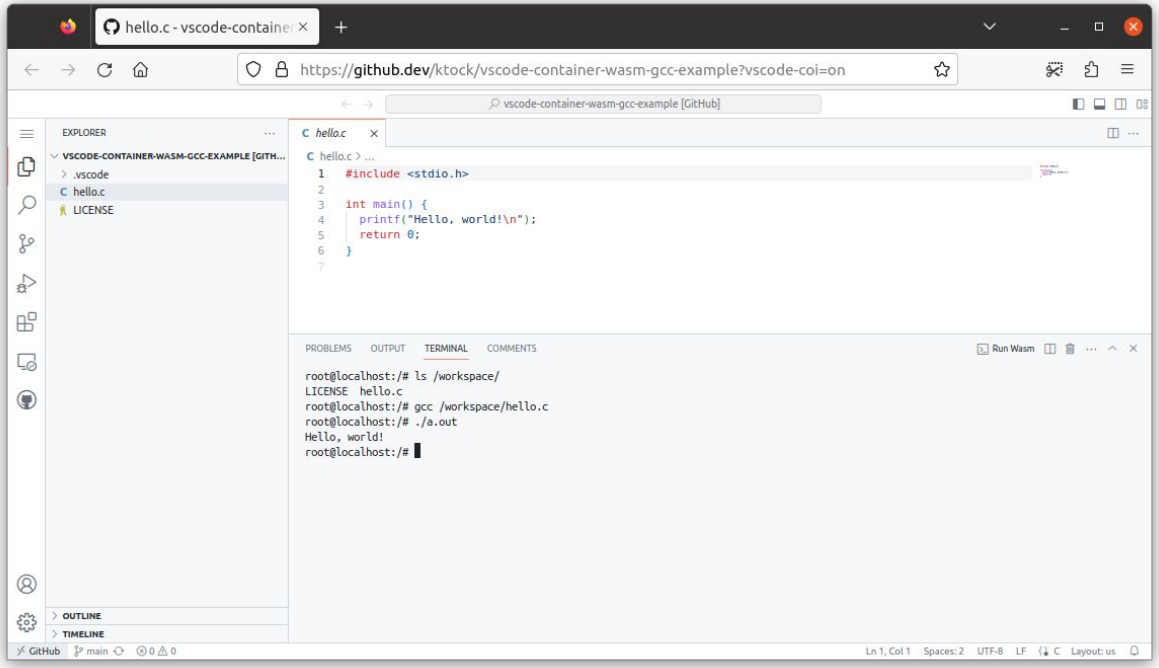
Summary

- Porting apps to Wasm costs time for re-compilation and/or re-implementation
- Container2wasm enables running unmodified containers on Wasm, leveraging CPU emulators
- Created an extension of VSCode for the Web to run containers on browser
- Future works: Performance analysis & improvement, CORS-enabled OS package repos, …

Debian container on browser



Debian container on `github.dev`



CLOUD NATIVE

Wasm DAY

NORTH AMERICA