vscode-container-wasm: An Extension of VSCode on Browser for Running Containers Within Your Browser

FOSDEM 2024 (Feb. 3)

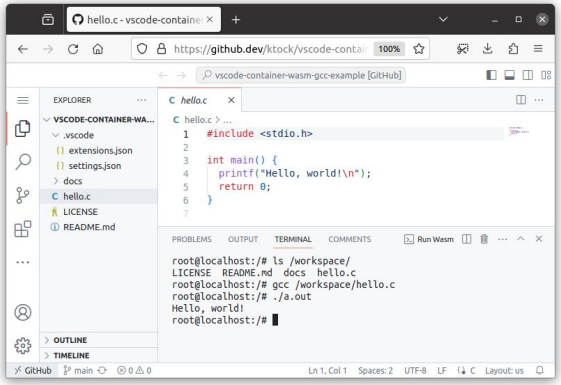
Kohei Tokunaga, NTT Corporation

Summary

- On-browser VSCode lacks Linux terminal running completely inside browser

- vscode-container-wasm enables to run Linux-based containers and its terminal **inside browser**

- Options for distributing containers to browsers
  - Pre-converting containers to Wasm images
  - Distributing OCI container images to browsers



Terminals on browser-based VSCode

On-browser VSCode lacks Linux terminal running completely inside browser

- Users can edit code but can't run it without remote machine
- Linux-based development tools (e.g. compilers) can't run in browser

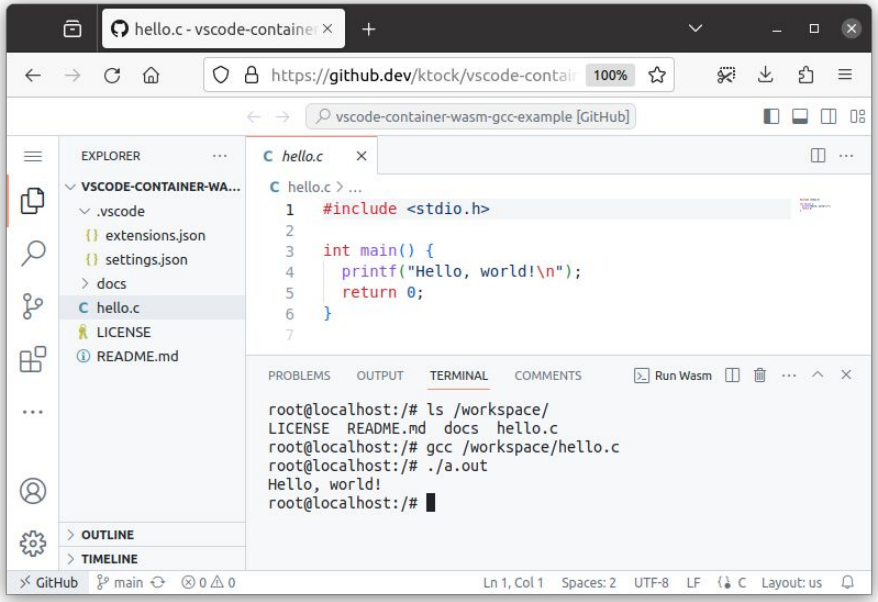
Browsers don’t provide Linux-compatible system

- Linux-based apps needs re-implementation to run inside browser
- Wasm lacks Linux-compatibility (e.g. no fork/exec)
- Can we run unmodified Linux terminal & dev environment inside browser?

vscode-container-wasm extension

`ktock.container-wasm`

- An experimental VSCode extension for running containers **inside browser**
- Workspace mounted at `/workspace/`
- HTTP(S) networking available w/ restrictions by browser (e.g. CORS)



How to distribute containers to browsers?

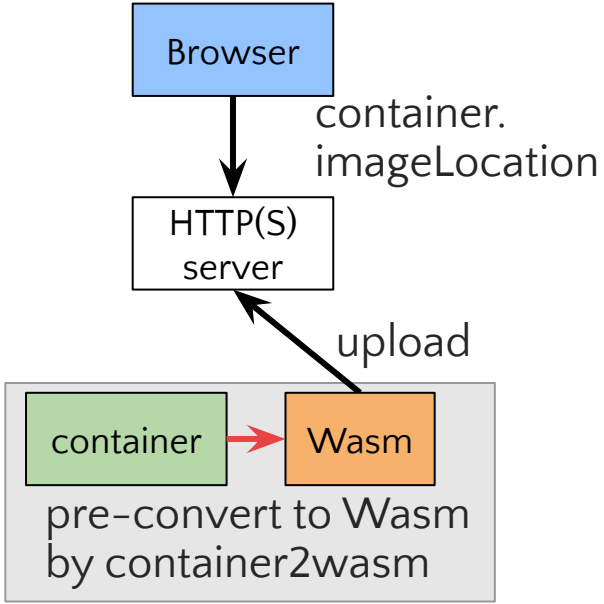
- Option A: Pre-converting containers to Wasm images
- Option B: Distributing OCI container images to browsers

Option A: Pre-converting containers to Wasm w/ container2wasm NTT

https://github.com/ktock/container2wasm

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

- An experimental Container to Wasm image converter
- Pros: container can run on non-browser VM (e.g. wasmtime) as well
- Cons: Pre-conversion is needed for each container to run

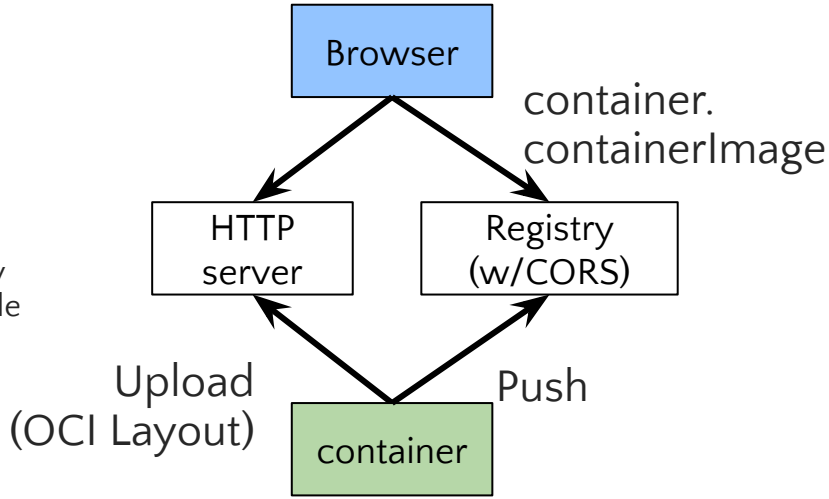


Example of `.vscode/settings.json`

```
{
   "container.imageLocation": "https://ktock.github.io/image-example/amd64-gcc-debian-wasi",
   "container.imageChunks": 9
}
```

Option B: Distributing OCI container image to browsers

- Registry needs to allow CORS access
  - But public registries doesn't allow CORS as of now
  - Try it on localhost registry with CORS header configured
    - https://github.com/ktock/container2wasm/blob/v0.6.2/extras/imagemounter/README.md#example-on-browser--registry
- Or, upload the container to HTTP(S) server in OCI Image Layout (files layout of `docker save` >= v25)



Example of `.vscode/settings.json`

Container registry

```
{"container.containerImage": "localhost:5000/ubuntu:22.04"}
```

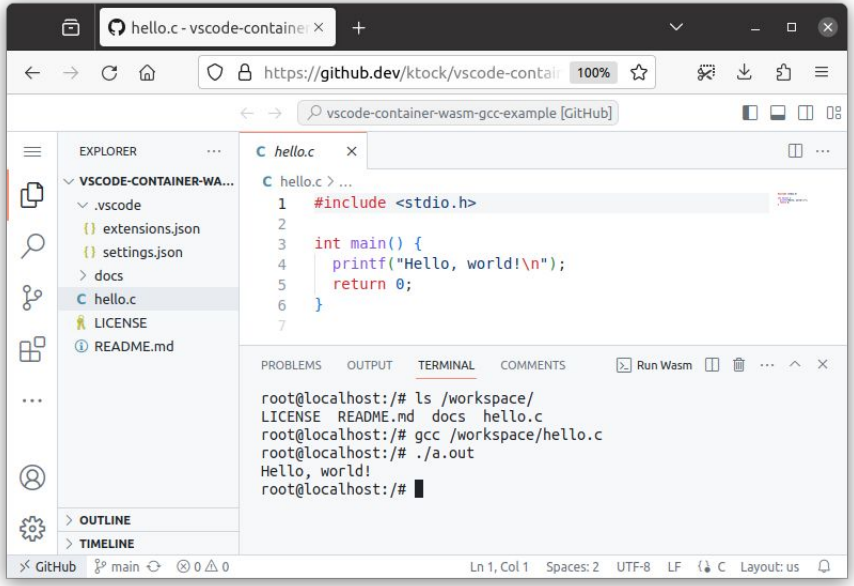
OCI Image Layout over HTTPS

```
{"container.containerImage": "https://ktock.github.io/ubuntu-oci-images/ubuntu-22.04-org-amd64"}
```

Example: Containers on github.dev

```
FROM debian:sid-slim
RUN apt-get update && apt-get install -y gcc
```

- Container image: gcc on Debian
- Workspace mounted at `/workspace/`
- HTTP(S) networking is available



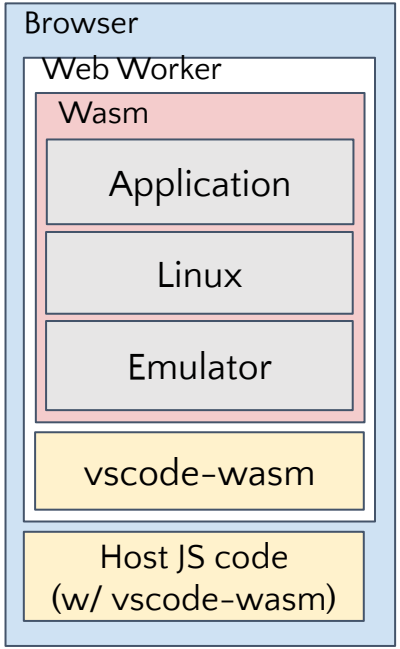
https://github.dev/ktock/vscode-container-wasm-gcc-example?vscode-coi=on

Demo

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

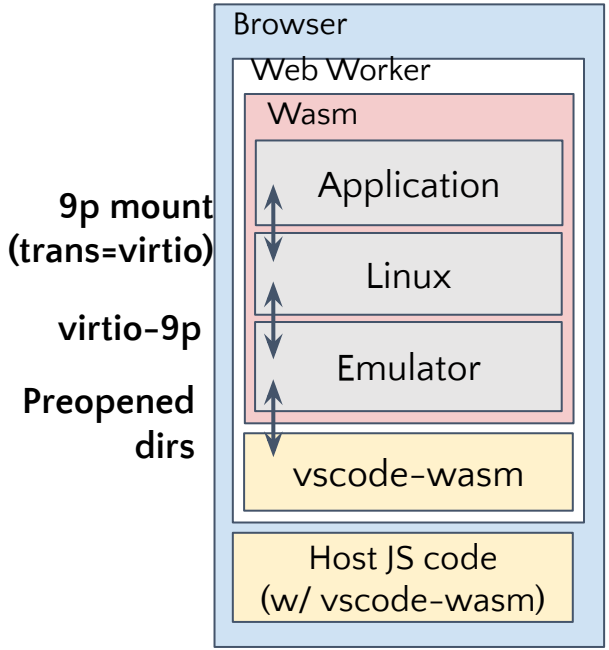
How it works

- Container and Linux run inside Wasm VM
- Utilizes CPU emulator
  - Bochs (for x86_64 containers)
  - TinyEMU (for RISC-V containers)
- Uses microsoft/vscode-wasm for Wasm/Wasi host



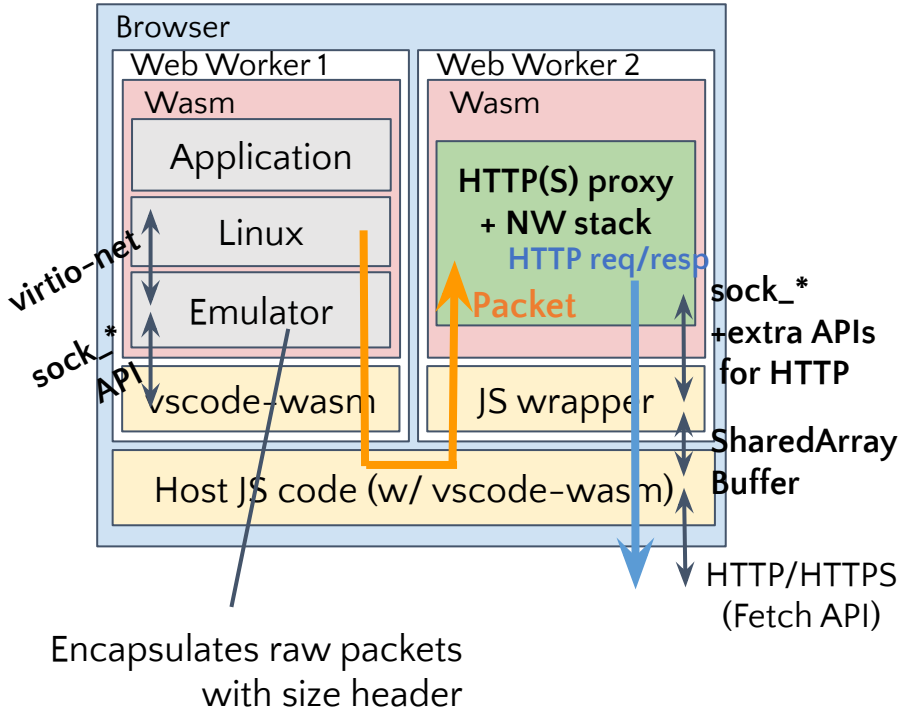
How it works: Mounting workspace to containers

- Workspaces are provided as pre-opened (mapped) dirs to Wasm VM
- Emulator shares pre-opened (mapped) dirs to the guest via virtio-9p



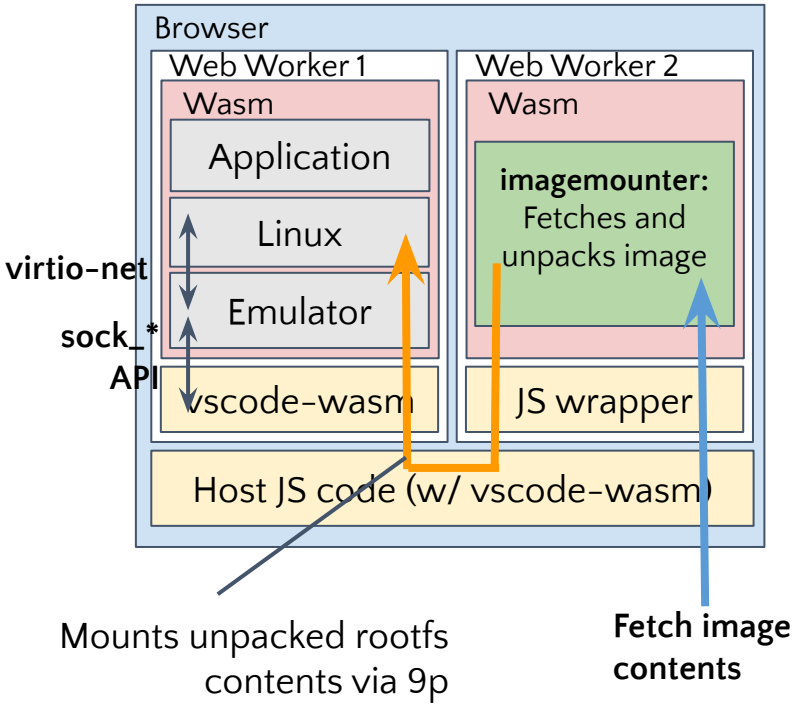
How it works: Networking

- NW stack + HTTP(S) proxy runs **inside browser**
- Forwards HTTP(S) using Fetch API
- Restrictions by Fetch API
  - Accessible sites limited by CORS
  - Forbidden Headers uncontrollable



How it works: Pulling containers to browser

- Container is fetched and unpacked **inside browser**
- Rootfs is mounted to the guest via 9p



Enabled only when "**Option B**" (Distributing OCI container images to browsers) is configured

Possible use cases of containers on Wasm

- Interactive on-browser linux-based demo
- On-browser development and testing
- Sandboxed execution environment of containers
- Application debugger runnable on browser
- Record & Replay debugging
- etc...

Related works

VMs on browser

- **v86**: https://github.com/copy/v86
  - x86-compatible on-browser CPU emulator by Fabian Hemmer
  - Supports wide variety of guest OSes (including Windows)
  - No support for x86_64

- **TinyEMU**: https://bellard.org/tinyemu/
  - RISC-V and x86 emulator by Fabrice Bellard
  - Can run on browser
  - Container2wasm uses this for RISC-V emulation
  - No support for x86_64

Future works

- Performance analysis & improvement
  - Possible integration with **elfconv**: https://github.com/yomaytk/elfconv
    - AOT compiler of Linux/aarch64 ELF to Wasm by Masashi Yoshimura, NTT Corporation
    - **Check also 16:00, Feb 4 @ LLVM devroom**: "elfconv: AOT compiler that translates Linux/AArch64 ELF binary to LLVM bitcode targeting WebAssembly", by Masashi Yoshimura, NTT Corporation

- Integration of container ecosystem with browsers
  - Accessing OS package repos (e.g. apk, apt, fedora packages, ...) from browser
  - CORS-allowed container registries

- Graphics support

Summary

- On-browser VSCode lacks Linux terminal running completely inside browser

- vscode-container-wasm enables to run Linux-based containers and its terminal **inside browser**

- Options for distributing containers to browsers
  - Pre-converting containers to Wasm images
  - Distributing OCI container images to browsers