



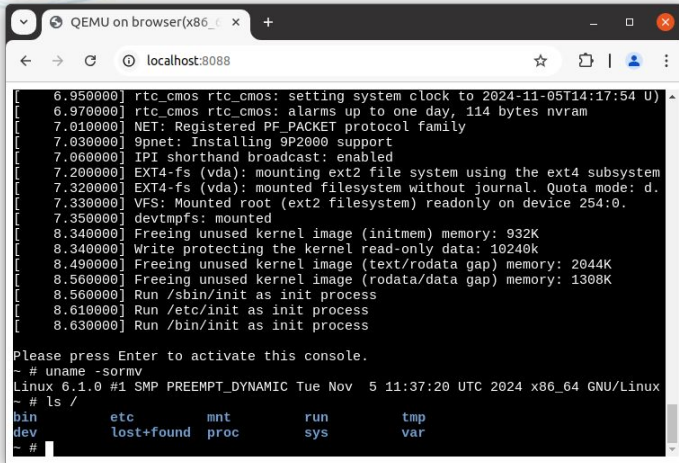
Running QEMU Inside Browser

FOSDEM 2025 (Feb. 2)

Kohei Tokunaga, NTT Corporation

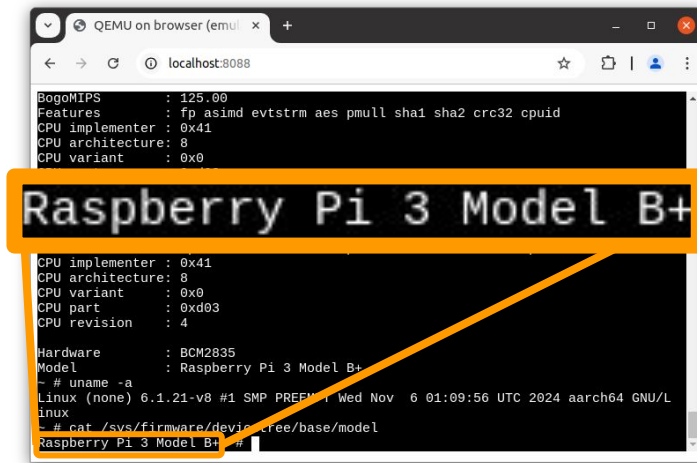
Summary

- QEMU Wasm is QEMU experimentally ported to browser
 - Runs unmodified softwares (e.g. Linux) inside browser
 - Supports TCG(JIT compiler), networking and mount
- Demos
 - Linux VM, containers, Raspberry Pi inside browser



```
6.950000] rtc_cmos rtc_cmos: setting system clock to 2024-11-05T14:17:54 U)
6.970000] rtc_cmos rtc_cmos: alarms up to one day, 114 bytes nvram
7.010000] NET: Registered PF_PACKET protocol family
7.030000] spmst: Installing gp2000 support
7.060000] IPI shorthand broadcast: enabled
7.200000] EXT4-fs (vda): mounting ext2 file system using the ext4 subsystem
7.320000] EXT4-fs (vda): mounted filesystem without journal. Quota mode: d.
7.330000] VFS: Mounted root (ext2 filesystem) readonly on device 254:0.
7.350000] devtmpfs: mounted
8.340000] Freeing unused kernel image (initmem) memory: 932K
8.340000] Write protecting the kernel read-only data: 10240k
8.490000] Freeing unused kernel image (text/rodata gap) memory: 2044K
8.560000] Freeing unused kernel image (rodata/data gap) memory: 1308K
8.560000] Run /sbin/init as init process
8.610000] Run /etc/init as init process
8.630000] Run /bin/init as init process

Please press Enter to activate this console.
~ # uname -sormv
Linux 6.1.0 #1 SMP PREEMPT_DYNAMIC Tue Nov 5 11:37:20 UTC 2024 x86_64 GNU/Linux
~ # ls /
bin          etc          mnt          run          tmp
dev          lost+found  proc         sys          var
~ #
```



```
BogoMIPS      : 125.00
Features      : fp asimd evtstrm aes pmull sha1 sha2 crc32 cpuid
CPU implementer : 0x41
CPU architecture: 8
CPU variant   : 0x0
CPU part      : 0xd03
CPU revision  : 4

Hardware      : BCM2835
Model         : Raspberry Pi 3 Model B+
~ # uname -a
Linux (none) 6.1.21-v8 #1 SMP PREEMPT_DYNAMIC Wed Nov 6 01:09:56 UTC 2024 aarch64 GNU/Linux
~ # cat /sys/firmware/devicetree/base/model
Raspberry Pi 3 Model B+
~ #
```

Why porting apps to browsers?

- Leveraging existing apps on browser (dev environment, playground, building block, etc)
 - [Ruby.wasm](#)
 - [VSCode Python for the web](#)
 - [Sqlite3 on browser](#)
 - [Postgres on browser \(PGLite\)](#)
 - [Swift on browser](#)
 - [Clang in browser](#)

But, porting apps to browsers is hard

- Existing softwares (e.g. Linux apps) need re-implementation to run inside browsers
 - Recompilation to Wasm (or JS)
 - Some syscalls (e.g. fork/exec) might be unavailable
- Can we run unmodified applications inside browsers?



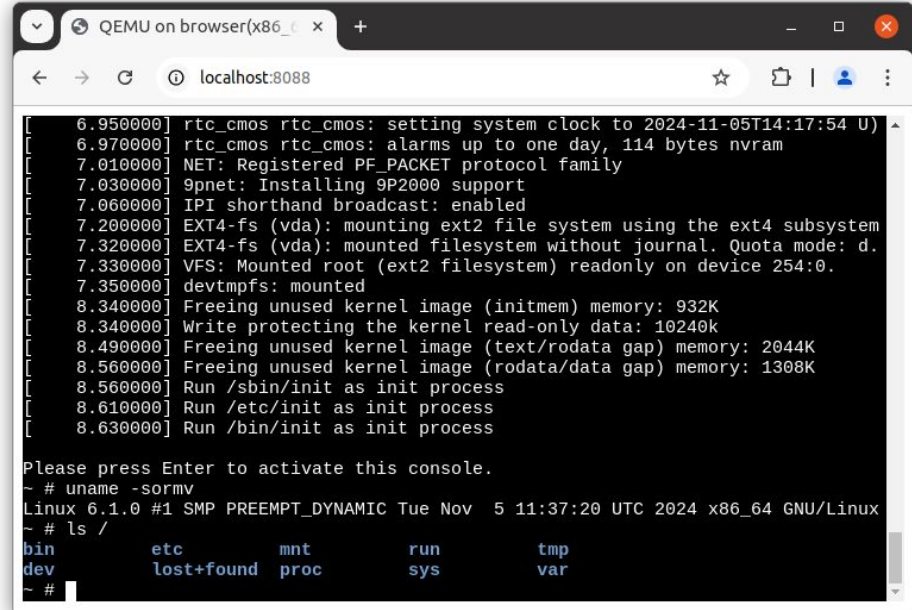
QEMU Wasm

QEMU Wasm

<https://github.com/ktock/qemu-wasm>

- Experimentally ported QEMU to browser, using emscripten
- Supports x86_64, AArch64, RISCv64 guests
- Supports TCG(JIT compiler), mount and networking, etc.

Example: x86_64 Linux on browser



The screenshot shows a browser window titled "QEMU on browser(x86_64)" with the address bar at "localhost:8088". The main content is a terminal window displaying the boot logs of a Linux system. The logs show the system clock being set to 2024-11-05T14:17:54 UTC, network initialization, and the mounting of the root filesystem. The prompt changes from a shell to a root shell (#) after the system has booted.

```
6.950000] rtc_cmos rtc_cmos: setting system clock to 2024-11-05T14:17:54 U)
6.970000] rtc_cmos rtc_cmos: alarms up to one day, 114 bytes nvram
7.010000] NET: Registered PF_PACKET protocol family
7.030000] 9pnet: Installing 9P2000 support
7.060000] IPI shorthand broadcast: enabled
7.200000] EXT4-fs (vda): mounting ext2 file system using the ext4 subsystem
7.320000] EXT4-fs (vda): mounted filesystem without journal. Quota mode: d.
7.330000] VFS: Mounted root (ext2 filesystem) readonly on device 254:0.
7.350000] devtmpfs: mounted
8.340000] Freeing unused kernel image (initmem) memory: 932K
8.340000] Write protecting the kernel read-only data: 10240k
8.490000] Freeing unused kernel image (text/rodata gap) memory: 2044K
8.560000] Freeing unused kernel image (rodata/data gap) memory: 1308K
8.560000] Run /sbin/init as init process
8.610000] Run /etc/init as init process
8.630000] Run /bin/init as init process

Please press Enter to activate this console.
~ # uname -srmv
Linux 6.1.0 #1 SMP PREEMPT_DYNAMIC Tue Nov 5 11:37:20 UTC 2024 x86_64 GNU/Linux
~ # ls /
bin          etc          mnt          run          tmp
dev          lost+found  proc         sys          var
~ #
```

Configuring flags

- Pass flags to QEMU via emscripten's **Module** object in JS
- [In the repo](#), examples are available for NW (**-netdev**), mount (**-virtfs**), migration (**-incoming**) flags

Example configuration

```
Module['arguments'] = [  
  '-nographic', '-m', '512M', '-accel', 'tcg,tb-size=500',  
  '-L', '/pack/',  
  '-drive', 'if=virtio,file=/pack/rootfs.bin',  
  '-kernel', '/pack/bzImage',  
  '-append', 'console=ttyS0 root=/dev/vda',  
];
```

Demo

- x86_64 Alpine Linux inside browser
- Demo page: <https://ktock.github.io/qemu-wasm-demo/>



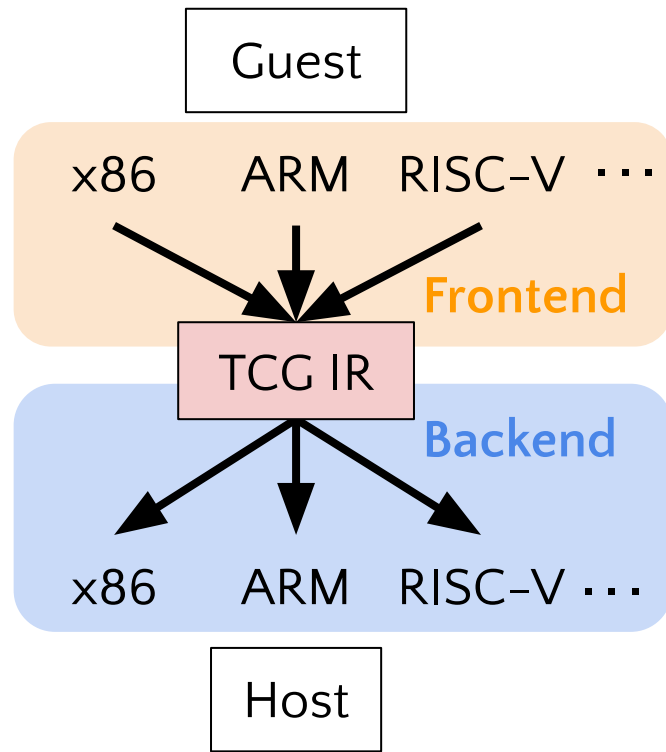
How it works?

How it works?

- QEMU (qemu-system-*) is compiled using emscripten
- Dependencies (e.g. kernel and rootfs) are packaged using emscripten's `--preload`
- Relies on browser APIs for JIT and networking (described later)

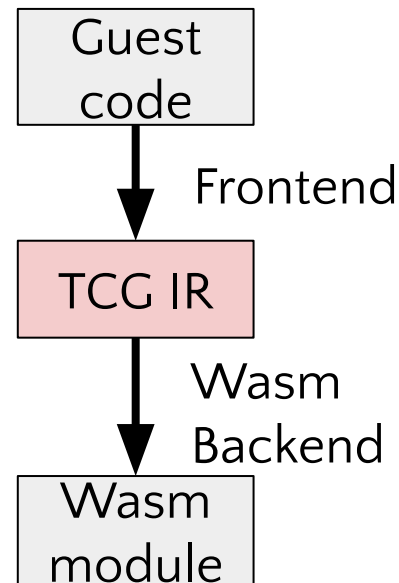
QEMU TCG (Tiny Code Generator)

- JIT binary translator of QEMU
- IR: Intermediate Representation
 - **Frontend** translates guest binaries to IR
 - **Backend** translates IR to the host arch
- Utilizes multi cores with MTTCG (Multi-Threaded TCG)



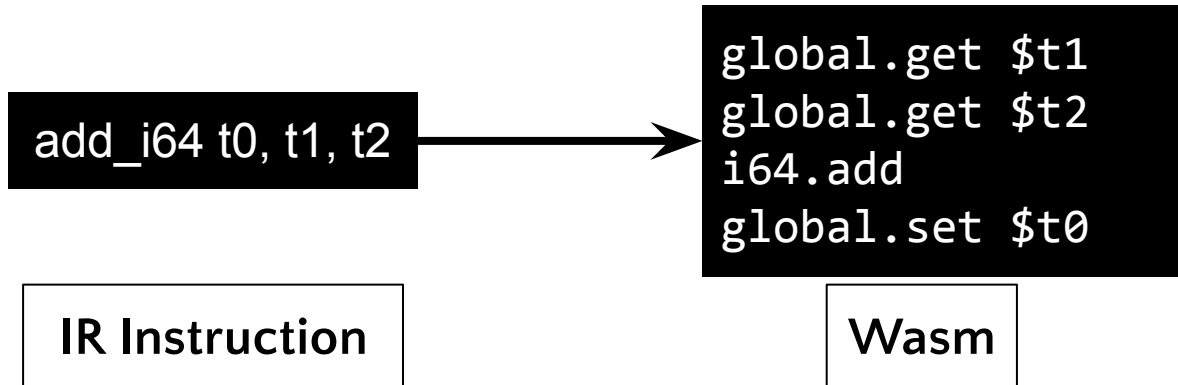
Wasm TCG backend

- Wasm can't execute code generated on memory
- Browser's APIs are used for compilation and execution
 - `WebAssembly.Module` compiles wasm code
 - `WebAssembly.Instance` makes it executable
 - Similar technique as used in other emulators e.g. [v86](#), [Qemu.js](#) (32bit guests and no multi-thread core though)
- Enabled emscripten's pthread to enable MTTCG



TCG IR to Wasm translation

- Translates each TB of IR to a Wasm module
 - TB=Translation Block; unit of instructions to translate
- Translates an IR instruction to Wasm instruction(s)
 - Added also 64bit IR instructions to enable 64bit guests and MTTCCG
- QEMU's memory and helper functions are imported to TB module



Mitigation for limitations of compiling modules

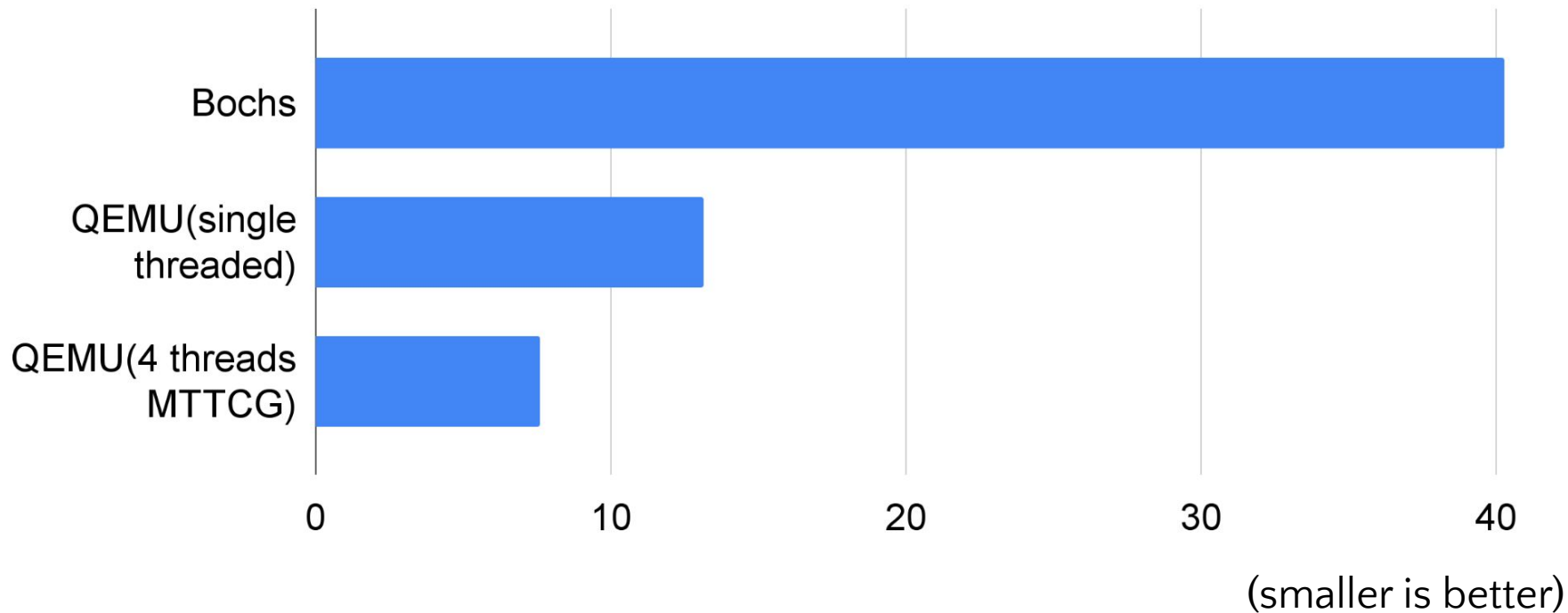
- Considerations for creating Wasm modules for each TB
 - Compilation overhead of modules
 - Browsers aren't capable of creating thousands of modules simultaneously
- Enabled both of TCI (built-in IR interpreter; slow) and Wasm backend
 - TBs run on TCI by default
 - TBs running many times (e.g. 1500) are compiled to Wasm

- Measured duration of compressing 10MB random data using pigz on emulated x86_64 guest
 - pigz is gzip implementation with multi processor support[1]
- Compared QEMU Wasm and Bochs ported to browser[2]
 - Bochs is a portable x86 emulator with interpreter approach
 - we've ported this to browser using emscripten
- Browser: Chrome 130.0.6723.58
- Host: Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz (8 cores)
- Script:
<https://github.com/ktock/container2wasm/commit/07260a2297ffc4ff40ca07dc6c558e4a8f56c154>

[1] <http://zlib.net/pigz/>

[2] <https://github.com/ktock/Bochs/tree/c2w-wasm>

pigz on emulators inside browser



Mounting filesystem

- Emscripten provides its own filesystem as **FS** API in JS
- QEMU Wasm can mount **FS** to the guest

Guest

```
$ mount -t 9p share0 /mnt/  
$ cat /mnt/file  
test
```

QEMU Wasm

```
-virtfs local,path=/share,mount_tag=share0...
```

JS (emscripten)

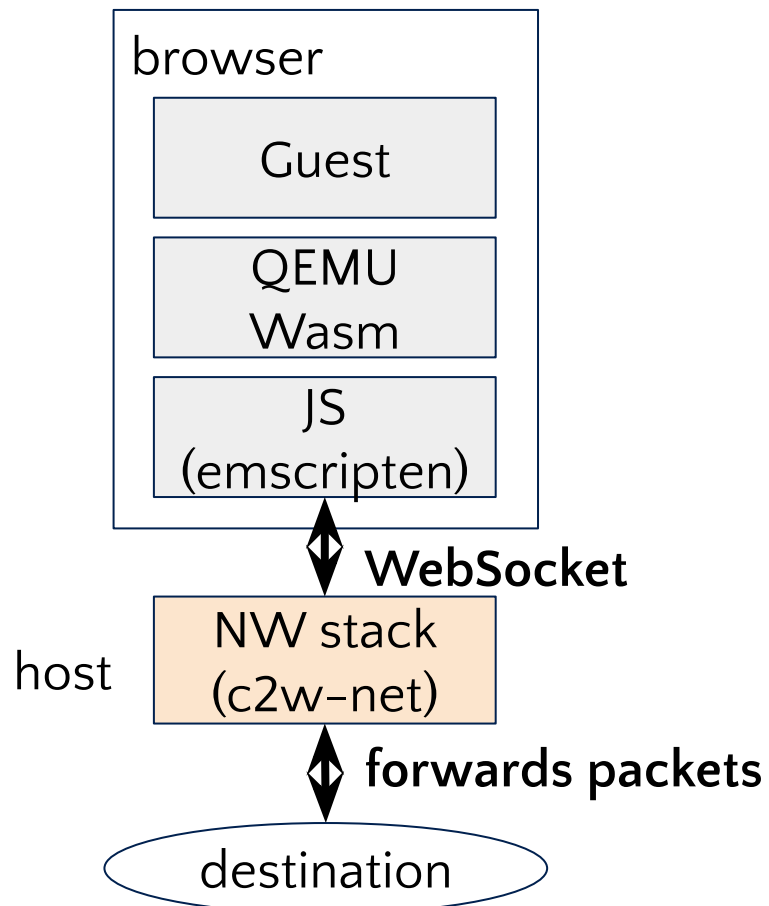
```
FS.writeFile('/share/file', 'test');
```

Two approaches are available

- **WebSocket-based approach**
 - Runs NW stack outside of browser
- **Fetch API-based approach**
 - Runs NW stack inside of browser

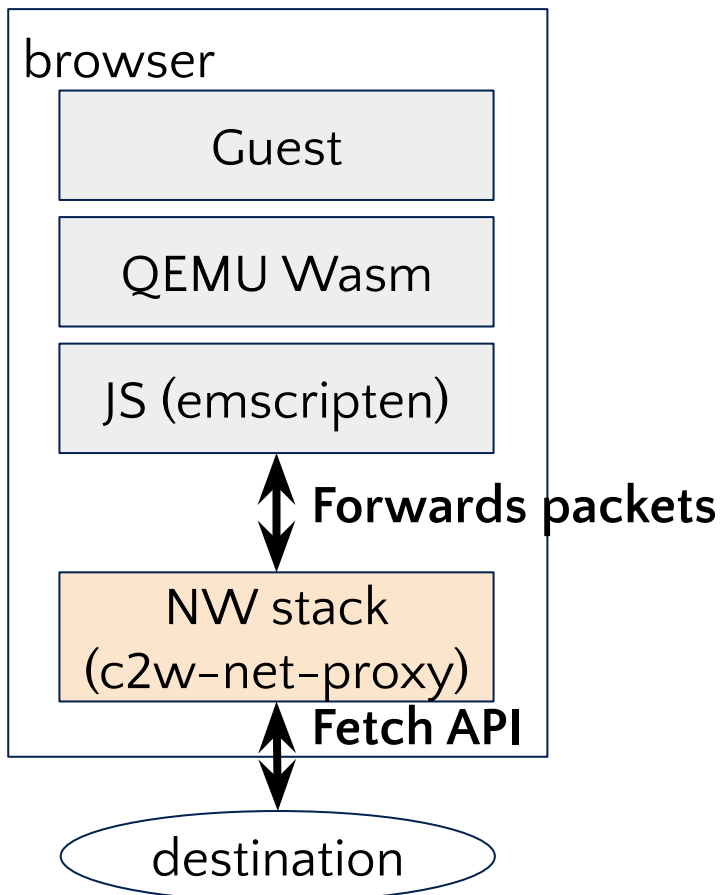
Networking utilizing WebSocket

- QEMU and the NW stack on the host are connected via WebSocket
- Pros: Destinations aren't limited by browser
- Cons: Maintenance cost of NW stack daemon on the host



Networking utilizing Fetch API

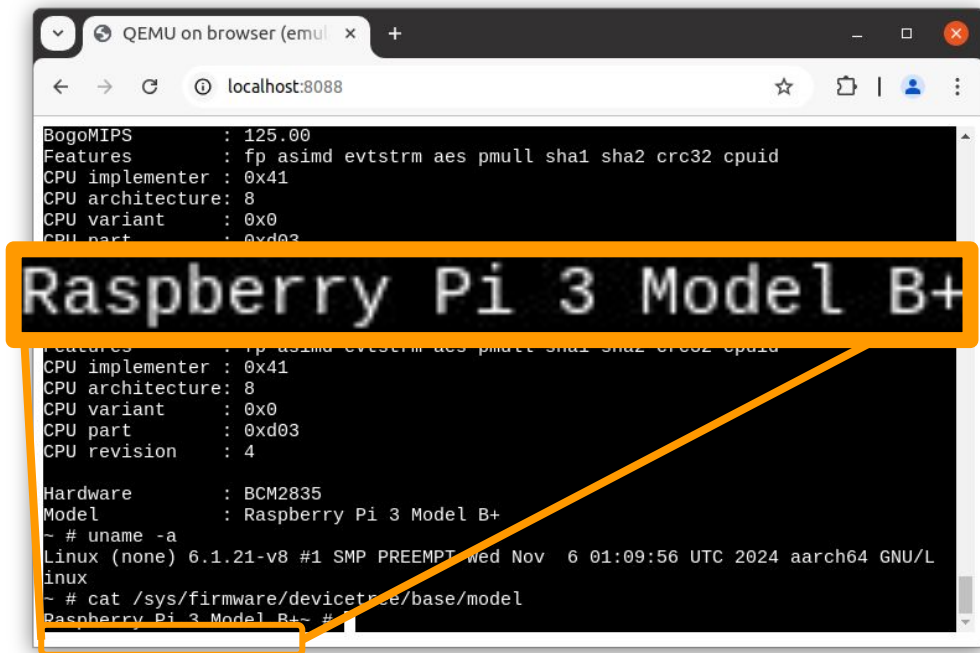
- NW stack inside browser proxies HTTP(S) connection using Fetch API
- Pros: Easy to maintain (no daemon on the host)
- Cons: HTTP(S) only. Restrictions by Fetch API
 - Limited destination by CORS
 - Forbidden Headers can't be controlled



Demos

Raspberry Pi emulation on browser

- Variety of machines (e.g. boards) are available thanks to QEMU
- Example: Raspberry Pi emulation



The screenshot shows a browser window titled "QEMU on browser (emul x)" with the address bar displaying "localhost:8088". The main content is a terminal window with a black background and white text. The terminal output includes system information for the emulated Raspberry Pi 3 Model B+:

```
BogoMIPS      : 125.00
Features      : fp asimd evtstrm aes pmull sha1 sha2 crc32 cpuid
CPU implemter : 0x41
CPU architect : 8
CPU variant   : 0x0
CPU part      : 0xd03

Raspberry Pi 3 Model B+

Features      : fp asimd evtstrm aes pmull sha1 sha2 crc32 cpuid
CPU implemter : 0x41
CPU architect : 8
CPU variant   : 0x0
CPU part      : 0xd03
CPU revision  : 4

Hardware     : BCM2835
Model        : Raspberry Pi 3 Model B+
~ # uname -a
Linux (none) 6.1.21-v8 #1 SMP PREEMPT Wed Nov  6 01:09:56 UTC 2024 aarch64 GNU/L
inux
~ # cat /sys/firmware/devicetree/base/model
Raspberry Pi 3 Model B+ #
```

An orange rectangular box highlights the text "Raspberry Pi 3 Model B+" in the terminal output. A yellow arrow points from the bottom right corner of this box to the bottom right corner of the terminal window.

Demo

- Raspberry Pi inside browser
- Demo page: <https://ktock.github.io/qemu-wasm-demo/>

Containers on browser with container2wasm

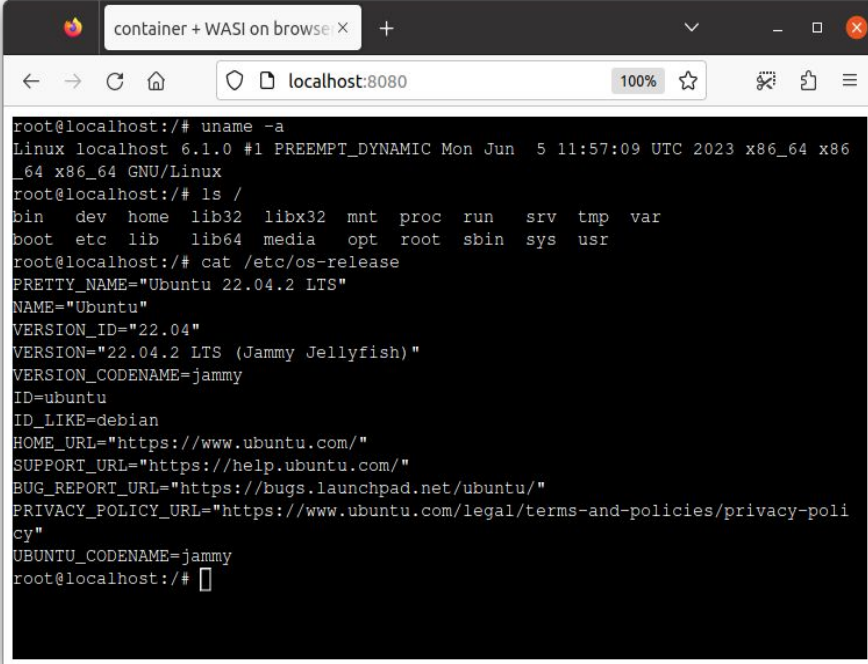
<https://github.com/ktock/container2wasm>

- container2wasm is a converter of a container to a Wasm blob
- Provides `--to-js` flag for enabling QEMU Wasm ($\geq v0.8$)

container

Wasm

```
$ c2w --to-js ubuntu:22.04 ./out/
```



```
container + WASI on browse x +
localhost:8080
root@localhost:/# uname -a
Linux localhost 6.1.0 #1 PREEMPT_DYNAMIC Mon Jun  5 11:57:09 UTC 2023 x86_64 x86_
64 x86_64 GNU/Linux
root@localhost:/# ls /
bin dev home lib32 libx32 mnt proc run srv tmp var
boot etc lib lib64 media opt root sbin sys usr
root@localhost:/# cat /etc/os-release
PRETTY_NAME="Ubuntu 22.04.2 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.2 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-poli
cy"
UBUNTU_CODENAME=jammy
root@localhost:/#
```


Demo

- Running a container inside browser
- Docs:

<https://github.com/ktock/container2wasm/tree/v0.8.0/examples/emscripten>

Future works

- **Performance & stability improvement for Wasm backend**
 - Still slower than other backends. Further improvement is needed.
- **Integration with more QEMU features**
 - More guest architectures, machines, graphics...
 - User mode QEMU
- **Integration with ecosystem**
 - Accessing package repos (e.g. apk, apt, ...) and container registries from browser (w/ CORS restriction)

- v86: <https://github.com/copy/v86>
 - x86-compatible on-browser CPU emulator by Fabian Hemmer
 - Supports wide variety of guest OSes (including Windows)
 - Supports JIT translation using browser APIs
 - No support for x86_64 guests
- Qemu.js: <https://github.com/atrosinenko/qemujs>
 - QEMU ported to browser by Anatoly Trosinenko
 - Supports JIT translation (TCG) using browser APIs
 - Single-threaded, no support for 64bit guests

Summary

- QEMU Wasm is QEMU experimentally ported to browser
 - Runs unmodified softwares (e.g. Linux) inside browser
 - Supports TCG(JIT compiler), networking and mount
- Demos
 - Linux VM, containers, Raspberry Pi inside browser

```
6.950000] rtc_cmos rtc_cmos: setting system clock to 2024-11-05T14:17:54 U
6.970000] rtc_cmos rtc_cmos: alarms up to one day, 114 bytes nvram
7.010000] NET: Registered PF_PACKET protocol family
7.030000] spnet: Installing gp2000 support
7.060000] IPI shorthand broadcast: enabled
7.200000] EXT4-fs (vda): mounting ext2 file system using the ext4 subsystem
7.320000] EXT4-fs (vda): mounted filesystem without journal. Quota mode: d
7.330000] VFS: Mounted root (ext2 filesystem) readonly on device 254:0.
7.350000] devtmpfs: mounted
8.340000] Freeing unused kernel image (initmem) memory: 932K
8.340000] Write protecting the kernel read-only data: 10240k
8.490000] Freeing unused kernel image (text/rodata gap) memory: 2044K
8.560000] Freeing unused kernel image (rodata/data gap) memory: 1308K
8.560000] Run /sbin/init as init process
8.610000] Run /etc/init as init process
8.630000] Run /bin/init as init process

Please press Enter to activate this console.
~ # uname -sormv
Linux 6.1.0 #1 SMP PREEMPT_DYNAMIC Tue Nov 5 11:37:20 UTC 2024 x86_64 GNU/Linux
~ # ls /
bin      etc      mnt      run      tmp
dev      lost+found  proc    sys      var
~ #
```

```
BogoMIPS      : 125.00
Features      : fp asimd evtstrm aes pmull sha1 sha2 crc32 cpuid
CPU implementer : 0x41
CPU architecture: 8
CPU variant   : 0x0
CPU part      : 0xd03
CPU revision  : 4

Hardware      : BCM2835
Model         : Raspberry Pi 3 Model B+
~ # uname -a
Linux (none) 6.1.21-v8 #1 SMP PREEMPT_DYNAMIC Wed Nov 6 01:09:56 UTC 2024 aarch64 GNU/L
~ # cat /sys/firmware/devicetree/base/model
Raspberry Pi 3 Model B+
~ #
```