WebAssembly, wasmCloud, and Containers

Reproducible macOS Benchmark Study (RSX207 Project 9)

Davi Khvedelidze

Tutors: Naresh Modina, Stefano Secci

January 9, 2026

Project Subject

- Edge and serverless systems are sensitive to cold-start latency and memory overhead.
- Containers are flexible but heavy for short-lived services.
- WebAssembly runtimes promise fast startup with smaller footprints.

Technological Context I

- Containers: Docker provides packaging and isolation but adds startup cost.
- WebAssembly: portable bytecode with near-native execution.
- wasmCloud: component model with capability providers and a host runtime.
- MicroVMs and unikernels are relevant but out of scope on macOS.

Technological Context II

- Wasmtime and WasmEdge run WASI modules directly with low overhead.
- wasmCloud adds a control plane (host + NATS + providers) for composition.
- Trade-off: raw startup and memory efficiency vs modularity and features.

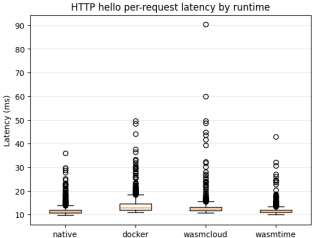
Base Specifications and Scope

- Workloads: hello-wasm, CPU hash, HTTP service with / and `/state`.
- Metrics: cold-start, warm latency, throughput, RSS, CPU utilization.
- Runtimes: native, Docker, wasmCloud, Wasmtime, WasmEdge on macOS M3 Pro.
- Firewall: macOS `pf` on/off to reflect real host policies.

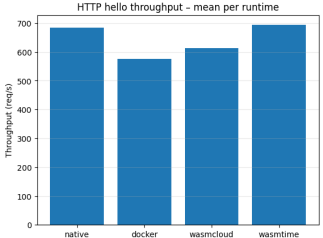
Methodology and Automation

- Cold start: launch to first HTTP 200 response.
- Warm runs: cached runtime and preloaded modules.
- HTTP runs: 5 warmups, 50 sequential latencies, 200 requests at concurrency 10.
- Full automation: scripts, raw logs, and plot generation per run.

HTTP Results: Latency and Throughput



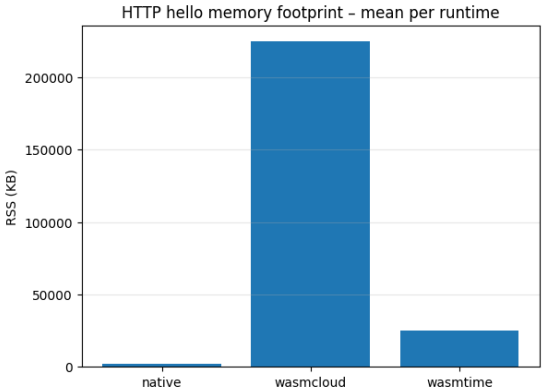
Latency distribution (ms)



Throughput (req/s)

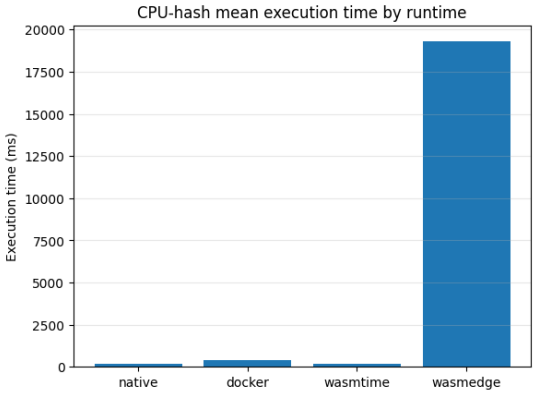
Takeaway: warm latency is similar across runtimes; throughput differences are modest.

HTTP Memory Footprint



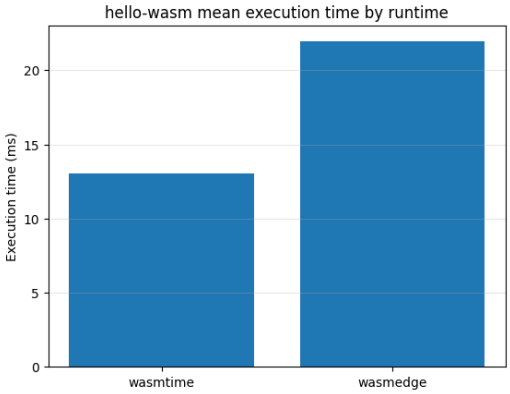
wasmCloud is highest because the measurement includes host, NATS, wadm, and the HTTP provider. Wasmtime is moderate, native is minimal.

CPU-Bound Results



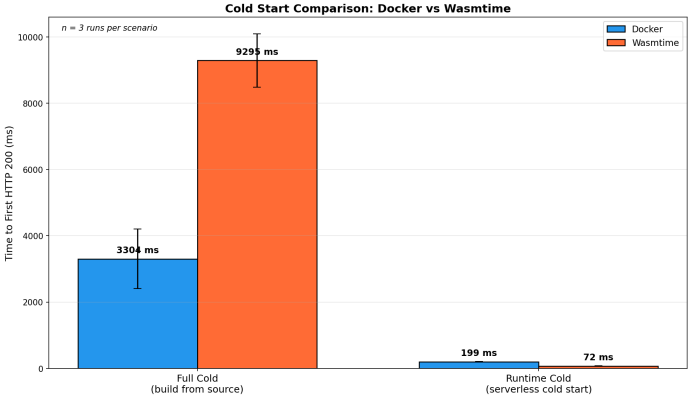
Wasmtime is close to native. Docker is slower on macOS. WasmEdge is much slower on this host, indicating platform-specific runtime differences.

Minimal WASI (hello-wasm)



This isolates invocation overhead. Wasmtime has lower fixed cost than WasmEdge on macOS, which matters for tiny serverless functions.

Cold-Start Comparison



Full cold is dominated by build time. Runtime cold shows Wasmtime starting faster than Docker.

Developments Achieved and Challenges

- Automated pipeline: build, run, log, and plot in one script.
- Firewall integration to measure realistic host constraints.
- wasmCloud readiness issues fixed with `wash up`, loopback binding, and cleanup.
- Reproducible datasets and timestamped plots for each campaign.

Summary and Next Steps

- Wasmtime delivers the fastest runtime cold starts with near-native CPU results.
- wasmCloud trades higher startup and memory cost for modularity and capabilities.
- Docker remains a stable baseline but has higher startup and CPU overhead.
- Next: Firecracker on Linux, richer workloads, and firewall-conditioned analysis.