Understanding & Evaluating WebAssembly Technologies For Serverless

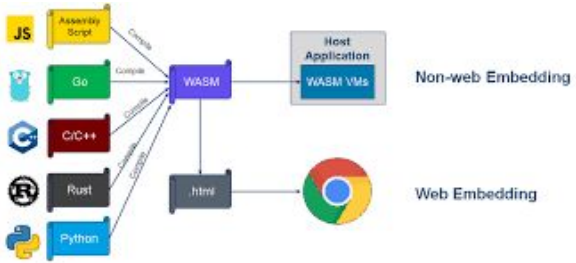
M Ali Ashraf Mohammad
Lawrence Wang

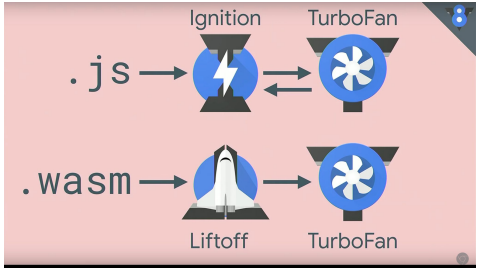
What is WebAssembly?

“WebAssembly (abbreviated Wasm) is a binary instruction format for a stack-based virtual machine.”

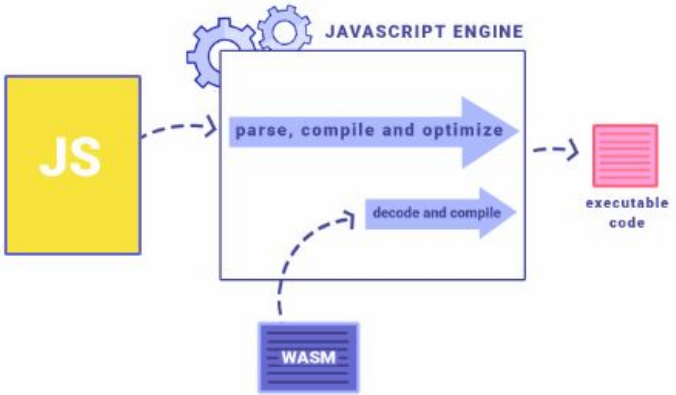
Wasm is designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications.

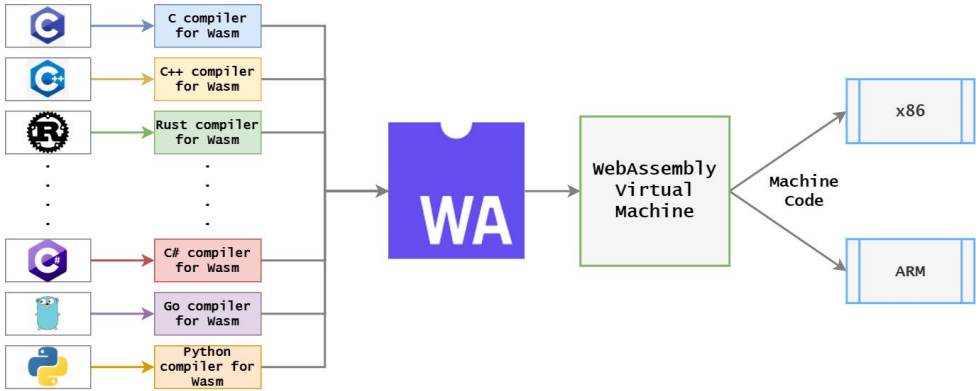
Why?





- What was it trying to solve?
- How it did it? - IR
- Being able to code in “any” programming language, compile it and run on the browsers.





Language support to wasm & usage

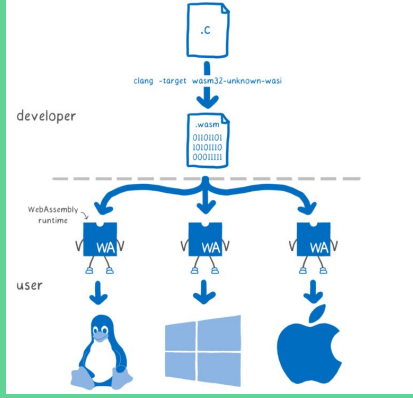
- WebAssembly runs in all major browsers and in all platforms. Developers can reasonably assume WebAssembly support is anywhere JavaScript is available.
- With around 40 languages that can compile to WebAssembly, developers can finally use their favorite language on the Web.

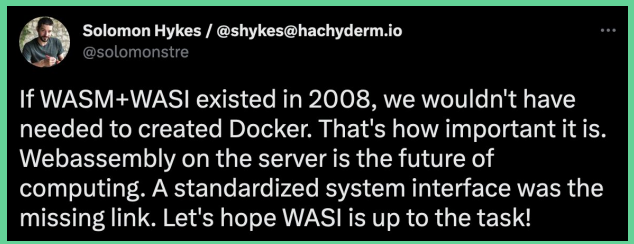
WebAssembly has been successfully deployed in the real world, too:

- eBay implemented a universal barcode scanner.
- Google Earth can run in any browser now, thanks to WebAssembly.
- The Doom 3 engine has also been ported to WebAssembly. You can play the demo online.
- Autodesk ported AutoCad to web browsers using WebAssembly.

Source: https://www.stackpath.com/

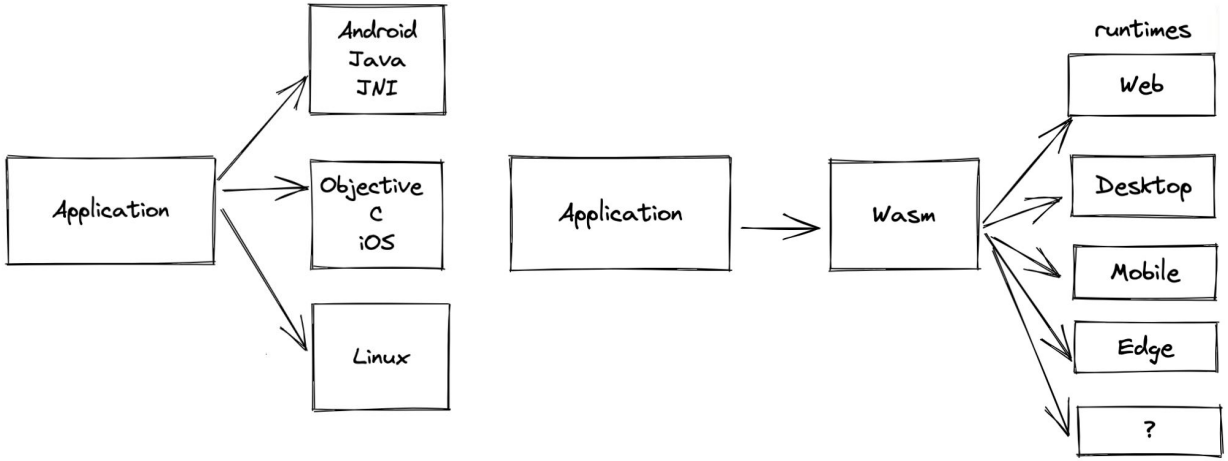
What is Wasm being used for now? - beyond the browser





Wasm future

- Compile once, run anywhere
- Mix and match “component” tools of any language
- Secure monolithic applications with plug-ins
- Serverless & FaaS



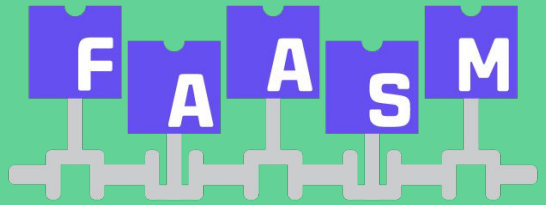
Wasm for serverless

- Current solution: Docker containers
  - Isolation overhead - ~100s milliseconds latency for containers
  - Relatively large memory footprint
  - Inefficient state sharing between containers
- WebAssembly
  - Safety guarantee via software fault isolation
  - Memory isolation with per process contiguous memory block allocation
- Problems to solve:
  - Lessen memory restrictions for state sharing
  - Provide interface for OS operations
  - Standardized orchestration

What is our objective?

- Understand the tradeoffs between different Wasm runtimes
- Evaluate performance metrics of platforms
  - Cold start time
  - Memory consumption
  - Network usage
  - Requests per second
- Verify the accuracy of Wasm performance advertisements
- Compare Wasm options with existing serverless technologies*
  - Knative
  - SPRIGHT
- Goal: build infrastructure around runtimes to extract metrics

*Broader objective of Abhishek Sharma's project



Flavours

- wasmer
- wasmtime
- lucet
- WAVM
- Second State VM
- V8 (nodeJS)
- dapr-wasm
- Intel WAMR
- wasm3

What are we evaluating?

Metrics

- CPU Utilisation, Cycles
- Memory Footprint
- Start-up times

Platforms

- Wasm Flavours (Sledge, Faasm, Wasmtime*).
- Existing serverless technologies (SPRIGHT & Knative).

Related Works

An Execution Model for Serverless Functions at the Edge

Adam Hall
Georgia Institute of Technology
Atlanta, Georgia
ach@gatech.edu

Umakishore Ramachandran
Georgia Institute of Technology
Atlanta, Georgia
rama@gatech.edu

A lightweight design for serverless Function-as-a-Service

Ju Long, Texas State University, Hung-Ying Tai, Shen-Ta Hsieh, and Michael Juntao Yuan, Second State LLC

Leaps and bounds: Analyzing WebAssembly's performance with a focus on bounds checking

Raven Szewczyk
University of Edinburgh
Edinburgh, UK
Raven.Szewczyk@ed.ac.uk

Kim Stonehouse
University of Edinburgh
Edinburgh, UK
Kim.Stonehouse@ed.ac.uk

Antonio Barbalace
University of Edinburgh
Edinburgh, UK
Antonio.Barbalace@ed.ac.uk

Tom Spink
University of St Andrews
St Andrews, UK
tcs6@st-andrews.ac.uk