WebAssembly (Wasm)

WebAssembly představuje čtvrtý pilíř webu (vedle HTML, CSS a JavaScriptu). Není to jazyk, který by lidé psali ručně, ale nízkoúrovňový binární formát, který prohlížeče dokáží zpracovat extrémní rychlostí. Je to otevřený standard vyvíjený pod hlavičkou W3C.

1. Proč WebAssembly vzniklo?

Tradiční webové aplikace spoléhají na JavaScript. Ten je sice výkonný díky JIT kompilaci v enginech jako V8, ale má své limity:

- **Parsování:** JS je textový formát, jehož parsování na mobilních zařízeních trvá dlouho.
- **Předvídatelnost:** Výkon JS může kolísat kvůli Garbage Collectoru a dynamickému typování.
- **Náročné výpočty:** Operace jako střih videa, 3D rendering nebo simulace jsou v JS neefektivní.

Wasm řeší tyto problémy tím, že nabízí binární kód, který je již optimalizován a připraven k okamžitému provedení.

2. Jak WebAssembly funguje?

Wasm neběží místo JavaScriptu, ale v úzké spolupráci s ním. Oba sdílejí stejné bezpečnostní „pískoviště“ (sandbox) prohlížeče.

Proces nasazení:

```
1. **Kompilace:** Programátor napíše kód v [[Rust]]u nebo [[C++]] a pomocí
[[LLVM]] jej zkompiluje do souboru ''.wasm''.
2. **Načtení:** Prohlížeč stáhne binární soubor.
3. **Validace a JIT:** Prohlížeč bleskově ověří bezpečnost kódu a přeloží
jej do strojového kódu daného procesoru ([[CPU|x86]] nebo [[CPU|ARM]]).
4. **Běh:** Kód běží v izolovaném paměťovém prostoru.
```

3. Klíčové vlastnosti

- **Nativní rychlost:** Wasm využívá schopnosti moderního hardwaru a dosahuje 80–90 % výkonu nativní aplikace.
- **Bezpečnost:** Běží v bezpečném sandboxu, nemá přímý přístup k souborovému systému ani k hardwaru (pokud mu to prohlížeč explicitně nepovolí).
- **Kompaktnost:** Binární formát je mnohem menší než ekvivalentní kód v JavaScriptu, což zrychluje načítání stránek.
- **Vícejazyčnost:** Umožňuje přenést existující knihovny v C/C++ na web bez nutnosti jejich přepisu.

4. Architektura: Stack Machine

Wasm je navržen jako **virtuální zásobníkový stroj (stack machine)**. Instrukce berou hodnoty ze zásobníku a výsledky na něj vrací. To zjednodušuje verifikaci kódu a umožňuje velmi rychlou kompilaci do strojového kódu.

Formáty:

- **Binární (.wasm):** Kompaktní kód pro stroje.
- **Textový (.wat):** Čitelný pro člověka, používá S-výrazy (podobné Lispu). Slouží k ladění.

```
(module
  (func $add (param $a i32) (param $b i32) (result i32)
    local.get $a
    local.get $b
    i32.add)
  (export "add" (func $add))
)
```

5. WebAssembly mimo prohlížeč (WASI)

Wasm se díky své izolaci a přenositelnosti začíná prosazovat i na serverech a v edge computingu. K tomu slouží rozhraní **WASI (WebAssembly System Interface)**.

- **Cloud Native:** Wasm moduly startují 100x rychleji než Docker kontejnery.
- **Serverless:** Ideální pro krátkodobé funkce (FaaS), které musí být bezpečně izolovány.

6. Slavné případy užití

- **Adobe Photoshop:** Verze pro web běží díky Wasm (přenesený kód z C++).
- **Google Earth:** Původně desktopová aplikace, nyní dostupná v prohlížeči díky Wasm.
- **Figma:** Nástroj pro designéry využívá Wasm k vykreslování plátna s vysokým výkonem.
- **Unity / Unreal Engine:** Herní enginy umožňují export her přímo do webového prohlížeče.

—

> **Vztah k ostatním technologiím:** Zatímco Qt poskytuje nástroje pro tvorbu UI, WebAssembly umožňuje toto UI a jeho logiku spustit kdekoli. Spojení Rust + WebAssembly je dnes považováno za „zlatý standard“ pro bezpečný a rychlý webový vývoj.

Související: Rust, C++, JavaScript, LLVM, frontend