Comment ça marche

Dans cette partie, nous examinerons précisément les étapes pour passer du code source à son exécution en WebAssembly.

Étape 1 : Le code source

Dans un premier temps, nous devons écrire du code. Ce code peut être rédigé dans différents langages, à condition qu'ils soient compatibles avec un compilateur.

Il est également possible de partir d'un code existant, ce qui est plus fréquent lorsqu'on utilise C++, mais il est possible de le faire avec d'autres langages. Ces codes nécessitent généralement quelques adaptations, afin de se conformer aux contraintes des compilateurs.

Voici une liste non exhaustive des langages et leur maturité dans l'écosystème WebAssembly :

Les langages stables	**Les langages embryonnaires**
C#	Elexir
AssemblyScript	Java
C	Javascript
C++	Kotlin
F#	PHP
Go	Python
Zig	Ruby
Rust	Swift
WAT	Lisp

Étape 2 : La Compilation

Dans cette étape, nous allons utiliser un outil pour compiler notre code vers le WASM. Selon les cas, cet outil peut ajouter du code JavaScript qui entourera le code WebAssembly. Pour effectuer des actions qui sont aujourd'hui impossibles en WebAssembly, comme manipuler le DOM...

Certains compilateurs sont compatibles avec des interfaces de programmation. Par exemple, avec Emscripten, nous pouvons utiliser SDL et OpenGL.

Liste de compilateurs et des langages associés (liste non exhaustive) :

Compilateur	**Langage(s)**
Wasm-Pack	Rust
wat2wasm	WAT
Emscripten	C / C++
AsBuild	AssemblyScript
Zig	ZigLang
Blazor	C# / F#
TinyGo	Go

Nous avons énuméré les options les plus courantes, mais il existe parfois plusieurs manières de compiler du code source.

WASM Binaire

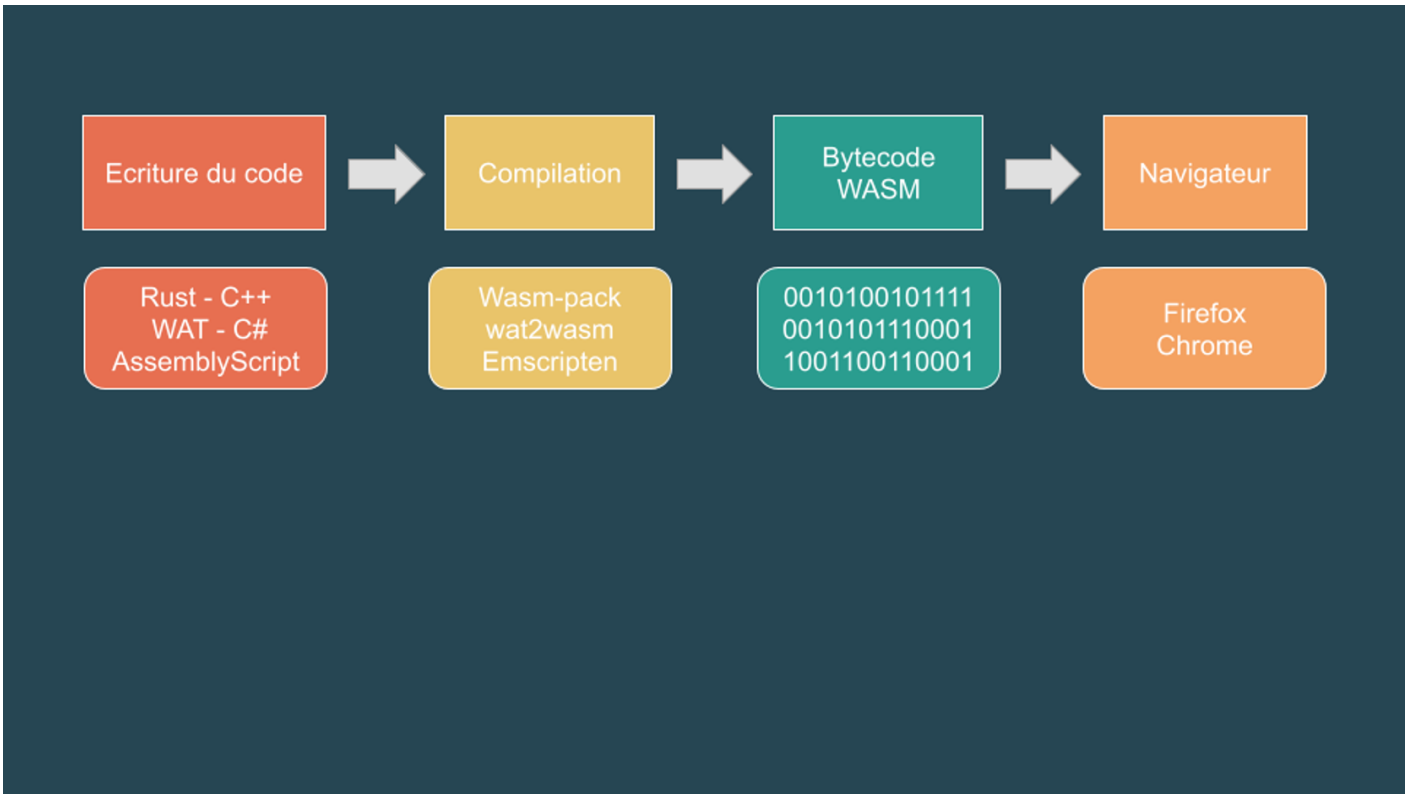
Une fois compilé, on se retrouve avec un ou des fichiers binaires. Ces fichiers ne sont pas lisibles par l'homme et contiennent les instructions du programme. Ces instructions, contrairement aux binaires classiques, ne sont pas destinées à des machines physiques, mais à des machines conceptuelles. C'est assez comparable au fonctionnement de la JVM. Une partie du travail est effectuée à la compilation, mais elle reste suffisamment floue pour pouvoir être interprétée par différentes architectures de processeur, différents navigateurs, etc.

Ce binaire est optimisé pour être rapide à charger, léger pour pouvoir être envoyé sur le réseau et pour s'exécuter dans un environnement isolé.

Interprétation

Une fois le fichier WASM créé, il ne nous reste plus qu'à l'exécuter. On peut le faire via un navigateur, mais il existe aussi d'autres solutions comme wasmtime ou des solutions cloud avec wasmedge et wasmer.

Ces solutions qui permettent d'exécuter du code WebAssembly côté serveur peuvent sembler contre-nature. Ne vous en faites pas, nous en reparlerons dans une section dédiée.



Revision #1
Created 2023-05-05 12:45:46 UTC by Noé Larrieu-Lacoste
Updated 2023-05-05 12:48:48 UTC by Noé Larrieu-Lacoste