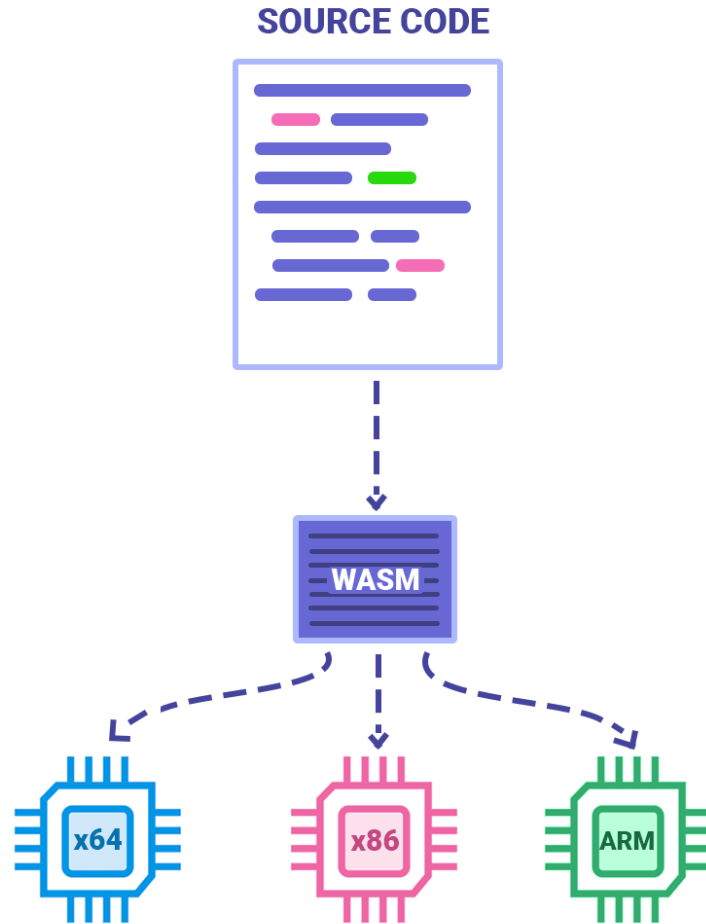


WebAssembly Dünyasında Neler Oluyor?

[Resmi sayfasındaki](#) tanıma bakacak olursak şu şekilde tanımlanmış.

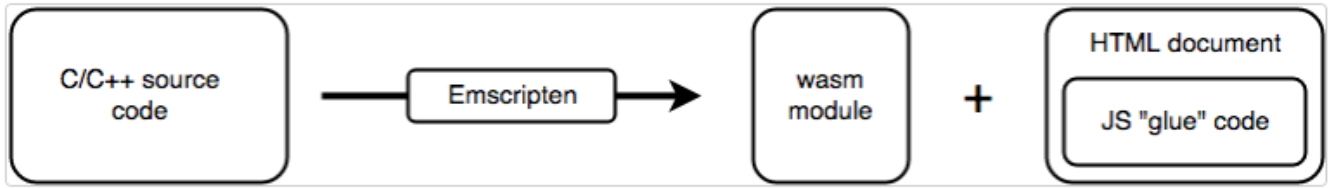
"WebAssembly (Wasm) stack-based virtual machine için tasarlanmış binary bir formattır. Hemen hemen bütün programlama dilleri için portable bir derleme hedefi olarak tasarlanmıştır. Hem client hem de server taraflı web uygulamaları geliştirebilmeyi sağlar."



Yani hemen hemen bütün programlama dilleriyle kod yazabileceğiz ve ortaya çıkan ürünü (wasm) çalıştırmak istediğimiz platform için bir runtime varsa çalıştırabileceğiz. Java'da bildiğiniz üzere bir çok platformda çalışabiliyor. Ancak WebAssembly Java'dan farklı olarak platform ile programlama dilini soyutluyor. Bu sayede istediğimiz programlama diliyle geliştirme yapabiliyoruz.

Her zaman sorulardan birini biz de soralım. Peki WebAssembly ilk çıkış itibariyle web browser'lar için çıktıysa bu JavaScript'in sonu mudur? Bu sorunun kısa cevabı **hayır**. O zaman ikisine ne gerek var diyebilirsiniz. Şimdi mevzuyu biraz açalım. Aslında hakikaten WebAssembly'nin **çıkış amacı** hızın önemli olduğu, yoğun hesaplama gereken yerlerde neredeyse native kadar hızlı çalışabilen bir yapı sunmaktı. Ancak bu kadar hızlı çalışabilen, taşınabilen, gömülü, modüler, dil bağımsız... vb. bir sistemi neden sadece web dünyasına sıkıştırılabilir ki diye düşünmüş olacaklar ki zamanla yeni özellikler ekleyerek kapsamını genişletmeye başladılar. İlk versiyonu Mart 2017'de çıkmış olmasına rağmen ve halen daha birinci versiyonu kullanılıyor olmasına rağmen belki de son zamanların en çok konuşulan, en hızlı gelişen teknolojilerden biri oldu diyebiliriz.

Bu gelişim sadece WebAssembly'i ekosistemin dışına çıkartmakla kalmadı aynı zamanda **ileride** JavaScript'e bir **alternatif** olma yolunda da etkiledi. Bunu da yine [resmi sayfasındaki](#) aşağıdaki mesajdan anlayabiliyoruz. Tabii ki bunun kısa sürede olmasını beklemek mantıklı olmayacaktır.



In a nutshell, the process works as follows:

1. Emscripten first feeds the C/C++ into clang+LLVM — a mature open-source C/C++ compiler toolchain, shipped as part of XCode on OSX for example.
2. Emscripten transforms the compiled result of clang+LLVM into a .wasm binary.
3. By itself, WebAssembly cannot currently directly access the DOM; it can only call JavaScript, passing in integer and floating point primitive data types. Thus, to access any Web API, WebAssembly needs to call out to JavaScript, which then makes the Web API call. Emscripten therefore creates the HTML and JavaScript glue code needed to achieve this.

Note: There are future plans to [allow WebAssembly to call Web APIs directly](#).

kaynak

Web API listesi için [Mozilla Developer Network](#) sayfasını ziyaret edebilirsiniz.

Sorumuza tekrar dönecek olursak WebAssembly'nin JavaScript'i ortadan kaldırma gibi bir hedefi yok. Zaten neden kaldırmayı da düşünsünler? Uzun yıllardır gösterdiği gelişimle kendini kanıtlamış, öğrenmesi ve kullanması çok kolay bir dil. WebAssembly'nin yaptığı veya yapacağı aslında sadece yeni oyunculara da kapıyı açmak olacaktır. Yani browser tarafı için bakacak olursak sadece JavaScript değil onunla birlikte diğer programlama dillerinin de kullanılabileceği bir ortam sunmaya çalışıyor WebAssembly.

Aslında teknolojik gelişmelere de bakacak olursak

- web 3.0'ın konuşulduğu,
- blockchain teknolojilerinin hızla geliştiği,
- yapay zekanın her alanda hayatımıza girdiği,
- modüler ve taşınabilir yapıların/teknolojilerin önem kazandığı,
- agility (çevikliğin) her sektörde konuşulduğu,
- her şeyin ucunun web teknolojilerine geçtiği

bir dünyada elimizdeki tek aracın JavaScript olması veya kalması da çok mantıklı görünmüyor. Böyle düşününce olur da WebAssembly bunu başaramazsa kesinlikle bir başka bir teknolojinin bu beklenen gelişimi göstereceğini tahmin etmek hiç de zor değil.

Bu teknolojinin geçen 5-6 yıllık bir geçmişi bize geleceği hakkında neler söylüyor ona bir bakalım.

İlk çıkış amacına baktığımızda web uygulamaları, hatta sadece client tarafında browser içinde çalışması için planlanmış olan WebAssembly geldiği noktada kendini çoktan aşmış görünüyor.

Yine [resmi sayfasında](#) yer alan kullanım alanlarına bakacak olursak alttaki uzun listeye ulaşıyoruz.

Browser İçinde Kullanım Alanları

- Better execution for languages and toolkits that are currently cross-compiled to the Web (C/C++, GWT, ...).
- Image / video editing.
- Games: (Casual games that need to start quickly, AAA games that have heavy assets, Game portals (mixed-party/origin content).
- Peer-to-peer applications (games, collaborative editing, decentralized and centralized).
- Music applications (streaming, caching).
- Image recognition.
- Live video augmentation (e.g. putting hats on people's heads).
- VR and augmented reality (very low latency).
- CAD applications.
- Scientific visualization and simulation.
- Interactive educational software, and news articles.
- Platform simulation / emulation (ARC, DOSBox, QEMU, MAME, ...).
- Language interpreters and virtual machines.
- POSIX user-space environment, allowing porting of existing POSIX applications.
- Developer tooling (editors, compilers, debuggers, ...).
- Remote desktop.
- VPN.
- Encryption.
- Local web server.
- Common NPAPI users, within the web's security model and APIs.
- Fat client for enterprise applications (e.g. databases).

Browser Dışında Kullanım Alanları

- Game distribution service (portable and secure).
- Server-side compute of untrusted code.
- Server-side application.
- Hybrid native apps on mobile devices.
- Symmetric computations across multiple nodes

Bahsi geçen alanlara alakalı bazı projelerin linkini paylaşıyorum.

- [Bu projede](#) WordPress'i PHO olmadan ve tamamen browser içinde çalıştırmayı başarmışlar. Git Projesine de [şu linkten](#) ulaşabilirsiniz.
- [Doom 3](#) Oyununu browser üzerinden oynayabilirsiniz.
- PostgreSQL veritabanını WebAssembly içinde çalıştırmayı başarmışlar. GitHub projesi için [şu linki](#) ziyaret ediniz.
- Adobe'nin yeni projesi, Photoshop'u browser üzerinden çalışmak için yaptığı çalışmayı [şu linkten](#) araştırabilirsiniz.

- Container teknolojileri ile ilgileniyorsanız Docker'ın WebAssembly tipindeki container çalışmasını incelemek için [şu linke](#) bakabilirsiniz. Bu konuyu makalenin devamında daha detaylı inceliyor olacağız.
- Linux, FreeBSD ve Windows işletim sistemlerinin WebAssembly ile browser içinde çalıştırıldığı projenin [web sayfası](#).
- x86 uyumlu CPU ve hardware'leri emulate eden ve makine kodlarını WebAssembly'ye runtime'da çeviren [bir çalışma](#). [Örnekleri](#) içinde işletim sistemlerinin farklı versiyonlarının browser içinden çalıştırıldığını görebilirsiniz.
- [FFmpeg](#)'i wasm üzerinden browser'da çalıştırabilirsiniz.
- [AutoCAD](#)'i web browser üzerinden kullanabilirsiniz.
- [Uno Platform](#) ile WebAssembly üzerinden cross platform uygulama geliştirebilirsiniz. Cross'dan kastım yazdığımız kodun sadece Android ve IOS değil, aynı zamanda Windows, Linux ve Mac'de de çalışması.

Daha birçok örneği internette bulmak mümkün.

WebAssembly'i çalıştıran web browser'lar birer runtime. WebAssembly'i browser dışında çalıştırmak için bir çok Runtime projesi mevcut. Bunların listesi için [şu Github sayfasına](#) bakabilirsiniz.

Github'da en çok yıldız alan bazı Runtime'lar

- **WasmEdge**

Ekosistemdeki en hızlı runtime'lardan biri. Halihazırda CNCF tarafından desteklenen bir proje.

C/C++, Rust, Go, Swift, Kotlin, AssemblyScript, Grain, JavaScript ve Python dillerini destekliyor.

İleride değineceğimiz Docker&Wasm ikilisinde de bu runtime kullanılmış.

Wasm runtime'ını yönetmek ve orkestra etmek için Kubernetes, blockhain,

GitHub Sayfası: <https://github.com/WasmEdge/WasmEdge>

- **Wasm3**

Python3, Rust, C/C++, GoLang, Zig, Perl, Swift, .Net, Nim, Arduino, PlatformIO, Particle, QuickJS dillerini destekliyor.

Embedded cihazlarda WebAssembly çalıştırılması üzerine yoğunlaşmış bir proje.

GitHub Sayfası: <https://github.com/wasm3/wasm3>

- **Wasmer**

Buraya yazamayacağım kadar çok dili destekliyor.

Run any code on any client. With WebAssembly and Wasmer.

Aynı zamanda WebAssembly Package Manager (WAPM) projesini de aynı community yönetiyor. İleride değineceğim için burada detaylarına girmiyoruz.

GitHub Sayfası: <https://github.com/wasmerio/wasmer>

- **Wasmtime**

C/C++, Rust, Go, Python, :net ve Go dillerini destekliyor.

GitHub Sayfası: <https://github.com/bytecodealliance/wasmtime>

Runtime'lar bunlarla sınırlı değil , daha fazlası için [şu sayfayı](#) ziyaret edebilirsiniz.

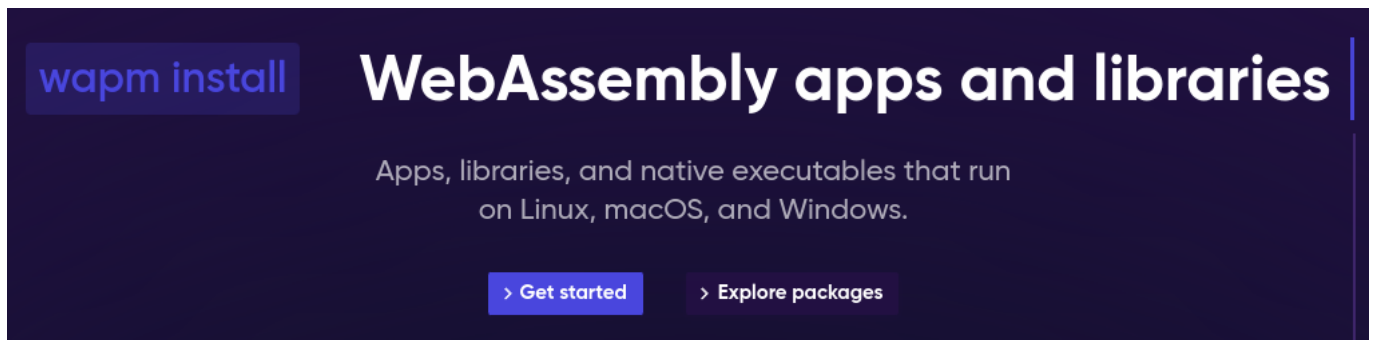
Diğer Araçlar

- **WASI (WebAssembly System Interface)**

[WASI](#) WebAssembly takımı tarafından geliştirilen we WebAssembly modüllerinin sistem kaynaklarına erişimini sağlayan bir middleware'dır.

WAPM (WebAssembly Package Manager)

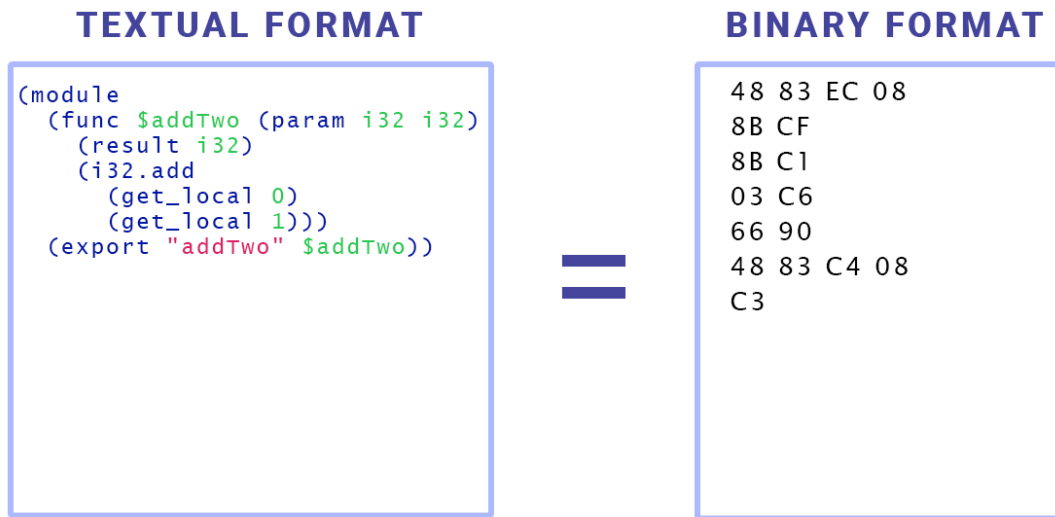
Wasmer runtime'ı geliştiren takımın geliştirdiği package menager aracı. Öncelikle sisteminize Wasmer runtime'ı ardından Wapm'ı kurmanız gerekiyor.



The banner features a dark blue background. On the left, there is a blue button with the text 'wapm install'. To its right, the text 'WebAssembly apps and libraries' is displayed in a large, white, sans-serif font. Below this, in a smaller white font, it says 'Apps, libraries, and native executables that run on Linux, macOS, and Windows.' At the bottom, there are two blue buttons: '> Get started' and '> Explore packages'.

<https://wapm.io/>

WebAssembly Teknik Detaylar



Text formatını uzantısı wat, binary formatın uzantısı wasm olarak kullanılır. Text formattan binary almak için alttaki komutu kullanabilirsiniz. Detaylar için [şu sayfayı](#) ziyaret edebilirsiniz.

```
wat2wasm simple.wat -o simple.wasm
```

Tersini yapmak da mümkün.

```
wasm2wat simple.wasm -o simple.wat
```

solomon hykes sözü

Solomon Hykes, a co-founder of Docker, wrote in 2019, "If WASM+WASI existed in 2008, we wouldn't have needed to create Docker. That's how important it is. WebAssembly on the server is the future of computing." [84] Wasmer, out in version 1.0, provides "software containerization, we create universal binaries that work anywhere without modification, including operating systems like Linux, macOS, Windows, and web browsers. Wasm automatically sandboxes applications by default for secure execution". [84]

kendi tweet i <https://twitter.com/solomonstre/status/1111004913222324225>

<https://en.wikipedia.org/wiki/WebAssembly>

mebbagistest.kizilay.org.tr (gui) mebbagistestapi.kizilay.org.tr 10.0.41.11 10.0.66.180

mebbagis.kizilay.org.tr (gui) bagismeb.kizilay.org.tr 10.0.40.23 10.0.6.180

Yapılmış Projeler

<https://make.wordpress.org/core/2022/09/23/client-side-webassembly-wordpress-with-no-server/>

<https://github.com/snaplet/postgres-wasm>

<https://github.com/wasmerio/wasmer-postgres>

<https://www.webassemblygames.com/>

<https://www.crazygames.com/game/zombsroyaleio>

web api ye erişebilmesi hedefine dair metin : <https://developer.mozilla.org/en-US/docs/WebAssembly/Concepts>

In a nutshell, the process works as follows:

1. Emscripten first feeds the C/C++ into clang+LLVM — a mature open-source C/C++ compiler toolchain, shipped as part of XCode on OSX for example.
2. Emscripten transforms the compiled result of clang+LLVM into a .wasm binary.
3. By itself, WebAssembly cannot currently directly access the DOM; it can only call JavaScript, passing in integer and floating point primitive data types. Thus, to access any Web API, WebAssembly needs to call out to JavaScript, which then makes the Web API call. Emscripten therefore creates the HTML and JavaScript glue code needed to achieve this.





 **Note:** There are future plans to [allow WebAssembly to call Web APIs directly](#).

The JavaScript glue code is not as simple as you might imagine. For a start, Emscripten implements popular C/C++ libraries like [SDL](#), [OpenGL](#), [OpenAL](#), and parts of [POSIX](#). These libraries are implemented in terms of Web APIs and thus each one requires some JavaScript glue code to connect WebAssembly to the underlying Web API.

peki web apiler nedir:

<https://developer.mozilla.org/en-US/docs/Web/API>

C

- [CSS Counter Styles](#)
- [CSS Font Loading API](#)
- [CSS Painting API](#)
- [CSS Properties and Values API](#) 
- [CSS Typed Object Model API](#)
- [CSSOM](#)
- [Canvas API](#)
- [Channel Messaging API](#)
- [Clipboard API](#)
- [Compression Streams API](#)
- [Console API](#)
- [Contact Picker API](#) 
- [Content Index API](#) 
- [Cookie Store API](#) 
- [Credential Management API](#)

D

- [DOM](#) 
- [Device Orientation Events](#)

E

- [Encoding API](#)
- [Encrypted Media Extensions](#)
- [EyeDropper API](#) 

F

- [Fetch API](#)
- [File API](#)
- [File System Access API](#)
- [File and Directory Entries API](#)
- [Fullscreen API](#)

G

- [Gamepad API](#)
- [Geolocation API](#)
- [Geometry Interfaces](#)

H

- [HTML DOM](#) 
- [HTML Sanitizer API](#) 
- [HTML Drag and Drop API](#)
- [History API](#)

Wasi (WebAssembly System Interface) Nedir

<https://github.com/WebAssembly/WASI>

Kesinlikle okunmalı: <https://hacks.mozilla.org/2019/03/standardizing-wasi-a-webassembly-system-interface/>

Bakılması gereken kaynaklar

- Bi bak: <https://hub.docker.com/r/renefonseca/wasisdk#/>

- makale : <https://www.thinktecture.com/en/webassembly/webassembly-with-dotnet/> uygulama : <https://github.com/thinktecture-labs/article-webassembly-dotnet-server>
- blazor uygulamasını browser dışında çalıştırılabilmek için detaylı anlatım: <https://dev.to/azure/exploring-net-webassembly-with-wasi-and-wasmtime-41l5>
- blockchain <https://medium.com/@rauljordan/webassembly-the-future-of-blockchain-computing-1a0ae28f7e40>
- blockchain uygulaması <https://www.codementor.io/@beber89/webassembly-to-run-blockchain-using-go-yuw6f9u7m>
- web 3 <https://101blockchains.com/web-3-0-blockchain-technology-stack/>

Kullanım Alanları

<https://webassembly.org/docs/use-cases/>

Browser İçinde Kullanım Alanları

- Better execution for languages and toolkits that are currently cross-compiled to the Web (C/C++, GWT, ...).
- Image / video editing.
- Games:
 - Casual games that need to start quickly.
 - AAA games that have heavy assets.
 - Game portals (mixed-party/origin content).
- Peer-to-peer applications (games, collaborative editing, decentralized and centralized).
- Music applications (streaming, caching).
- Image recognition.
- Live video augmentation (e.g. putting hats on people's heads).
- VR and augmented reality (very low latency).
- CAD applications.
- Scientific visualization and simulation.
- Interactive educational software, and news articles.
- Platform simulation / emulation (ARC, DOSBox, QEMU, MAME, ...).
- Language interpreters and virtual machines.
- POSIX user-space environment, allowing porting of existing POSIX applications.
- Developer tooling (editors, compilers, debuggers, ...).
- Remote desktop.
- VPN.
- Encryption.
- Local web server.
- Common NPAPI users, within the web's security model and APIs.
- Fat client for enterprise applications (e.g. databases).

Browser Dışında Kullanım Alanları

- Game distribution service (portable and secure).
- Server-side compute of untrusted code.
- Server-side application.
- Hybrid native apps on mobile devices.
- Symmetric computations across multiple nodes

Kaynaklar

- <https://hacks.mozilla.org/2019/03/standardizing-wasi-a-webassembly-system-interface/>
-