Cross-platform integration for Polyglot Programming

Feng Li (李枫)
hkli2013@126.com
Jan 14, 2023

Agenda

I. Background
- Tech Stack
- Testbed

II. .Net on JVM
- C# on GraalVM

III. Java on .Net
- Bytecode Conversion

IV. Wasm on JVM
- Wasm on GraalVM

V. Java on Wasm
- Bytecode Conversion

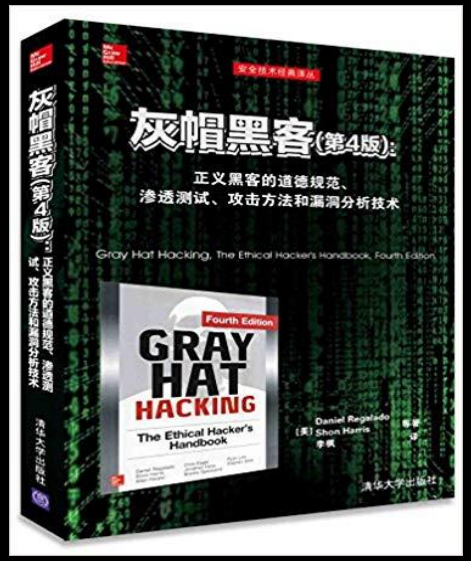
VI. .Net on Wasm
- ...

VII. Wasm on .Net
- Bytecode Conversion

X. Wrap-up

Who Am I

- An indie developer from China
- The main translator of the book «Gray Hat Hacking The Ethical Hacker's Handbook, Fourth Edition» (ISBN：9787302428671) & «Linux Hardening in Hostile Networks, First Edition» (ISBN: 9787115544384)



- Pure software developmement for ~15 years(~11 years on Mobile dev)
- Actively participate in various activities of the open source community
- https://github.com/XianBeiTuoBaFeng2015/MySlides/tree/master/Conf
  https://github.com/XianBeiTuoBaFeng2015/MySlides/tree/master/LTS
- Recently, focus on infrastructure of Cloud/Edge Computing, AI, Virtualization, Program Runtimes, Network, 5G, RISC-V, EDA…

I. Background

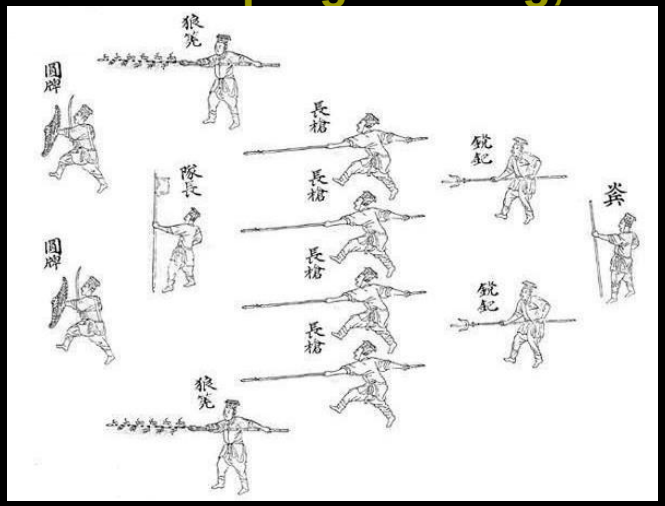
1) Tech Stack

1.1 Polyglot Programming

- https://en.wikipedia.org/wiki/Polyglot_(computing)

> In computing, a **polyglot** is a computer program or script written in a valid form of multiple programming languages or file formats.[1] The name was coined by analogy to multilingualism. A **polyglot file** is composed by combining syntax from two or more different formats.[2] When the file formats are to be compiled or interpreted as source code, the file can be said to be a **polyglot program**, though file formats and source code syntax are both fundamentally streams of bytes, and exploiting this commonality is key to the development of polyglots.[3] Polyglot files have practical applications in compatibility,[4] but can also present a security risk when used to bypass validation or to exploit a vulnerability.

- Use the best tool for the right jobs: high performance, scripting, web, functional programming, etc



- The most popular Polyglot runtimes: .Net, Wasm, GraalVM…
- …

1.2 Wasm

1.2.1 Overview

- https://en.wikipedia.org/wiki/WebAssembly

C source code and corresponding WebAssembly

C source code	WebAssembly . wat text format	WebAssembly . wasm binary format

`int factorial(int n) {`
`  if (n == 0)`
`  return 1;`
`  else`
`  return n * factorial(n-1);`
`}`
	`(func (param i64) (result i64)`
	`  local.get 0`
	`  i64.eqz`
	`  if (result i64)`
	`  i64.const 1`
	`  else`
	`  local.get 0`
	`  local.get 0`
	`  i64.const 1`
	`  i64.sub`
	`  call 0`
	`  i64.mul`
	`  end)`
		`00 61 73 6D 01 00 00 00`
		`01 00 01 60 01 73 01 73 06`
		`03 00 01 00 02`
		`0A 00 01`
		`00 00`
		`20 00`
		`50`
		`04 7E`
		`42 01`
		`05`
		`20 00`
		`20 00`
		`42 01`
		`7D`
		`10 00`
		`7E`
		`0B`
		`0B 15 17`

- https://webassembly.org/

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.

WebAssembly 1.0 has shipped in 4 major browser engines.

- https://github.com/WebAssembly/design
- https://webassembly.org/specs/
- https://blog.scottlogic.com/2022/06/20/state-of-wasm-2022.html
- https://platform.uno/blog/the-state-of-webassembly-2021-and-2022/
- https://www.w3.org/groups/wg/wasm
- ...

Implementations

- While WebAssembly was initially designed to enable near-native code execution speed in the web browser, it has been considered valuable outside of such, in more generalized contexts.[37][38] Since WebAssembly's runtime environments (RE) are low level virtual stack machines (akin to JVM or Flash VM) that can be embedded into host applications, some of them have found a way to standalone runtime environments like Wasmtime and Wasmer.[8][13]

  Web browsers [edit]

  In November 2017, Mozilla declared support "in all major browsers",[39] after WebAssembly was enabled by default in Edge 16.[40] The support includes mobile web browsers for iOS and Android. As of July 2021, 94% of installed browsers support WebAssembly.[41] But for older browsers, Wasm can be compiled into asm.js by a JavaScript polyfill.[42]

Compilers:

WebAssembly implementations usually use either ahead-of-time (AOT) or just-in-time (JIT) compilation, but may also use an interpreter. While the first implementations have landed in web browsers, there are also non-browser implementations for general-purpose use, including Wasmer,[10] Wasmtime[40] or WAMR,[16] wasm3, WAVM, and many others.[41]

Because WebAssembly executables are precompiled, it is possible to use a variety of programming languages to make them.[42] This is achieved either through direct compilation to Wasm, or through implementation of the corresponding virtual machines in Wasm. There have been around 40 programming languages reported to support Wasm as a compilation target.[43]

Emscripten compiles C and C++ to Wasm[26] using the Binaryen and LLVM as backend.[44] The Emscripten SDK can compile any LLVM-supported languages (such as C, C++ or Rust, among others) source code into a binary file which runs in the same sandbox as JavaScript code.[note 1] Emscripten provides bindings for several commonly used environment interfaces like WebGL.

As of version 8, a standalone Clang can compile C and C++ to Wasm.[49]

Its initial aim is to support compilation from C and C++,[50] though support for other source languages such as Rust, .NET languages[51][52][43] and AssemblyScript[53] (TypeScript-like) is also emerging. After the MVP release, there are plans to support multithreading and garbage collection[54][55] which would make WebAssembly a compilation target for garbage-collected programming languages like C# (supported via Blazor), F# (supported via Bolero[56] with help of Blazor), Python, and even JavaScript where the browser's just-in-time compilation speed is considered too slow. A number of other languages have some support including Python,[57] Julia,[58][59][60] and Ruby.[61]

- https://github.com/appcypher/awesome-wasm-langs
- https://github.com/appcypher/awesome-wasm-runtimes
- https://00f.net/2023/01/04/webassembly-benchmark-2023/

Limitations

1. In general, WebAssembly does not allow direct interaction with the DOM. All interaction must flow through JavaScript interop.
2. Absence of garbage collection (although there are plans to address this.)
3. Security considerations (discussed below)

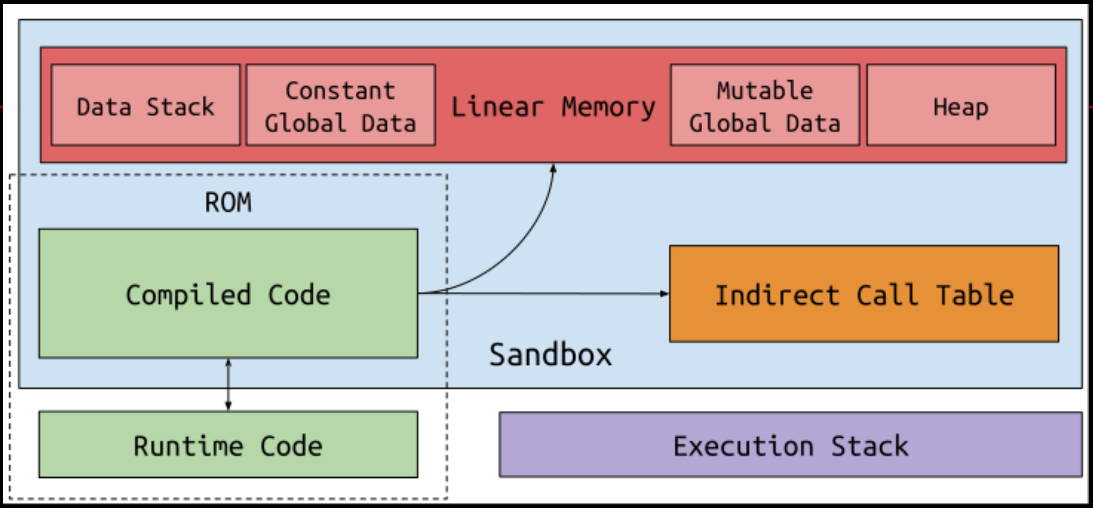
WebAssembly is supported on desktops, and mobile, but on the latter, in practice (for non-small memory allocations, such as with Unity game engine) there are "grave limitations that make many applications infeasible to be *reliably* deployed on mobile browsers [..] Currently allocating more than ~300MB of memory is not reliable on Chrome on Android without resorting to Chrome-specific workarounds, nor in Safari on iOS."[62]

There is no direct Document Object Model (DOM) access; however, it is possible to create proxy functions for this, for example through stdweb[63] or web_sys[64] when using the Rust language.

All major web browsers allow WebAssembly if Content-Security-Policy is not specified, or if "unsafe-eval" is used, but otherwise the major web browsers behave differently.[65] In practice WebAssembly can't be used on Chrome without "unsafe-eval",[66][67] while a worker thread workaround is available.[67]

1.2.2 Runtime

Sandboxing



Wasm uses a co-design between the compiler, and the dynamic checks of the runtime system to provide the sandbox that isolates the surrounding system from the logic of the contained code. The figure depicts the main aspects of the sandbox. These include:

- *Linear memory* that holds all memory accessed by the sandbox. The compiler emits code that checks that all loads and stores remain within the linear memory, thus preventing errant accesses outside the sandbox. Linear memory is expandable much like a traditional heap.
- The *indirect function call table* that facilitates function pointer calls. To ensure that function pointer invocations are safe (to code generated by the compiler), function pointers reference an *offset* into the table. Each entry includes the type of the function, and ensures that function invocations are well-typed.
- The separation of the *execution stack* -- used to track function calls -- and the *data stack* -- used to contain stack-allocated data that can be referenced, thus must be in linear memory.

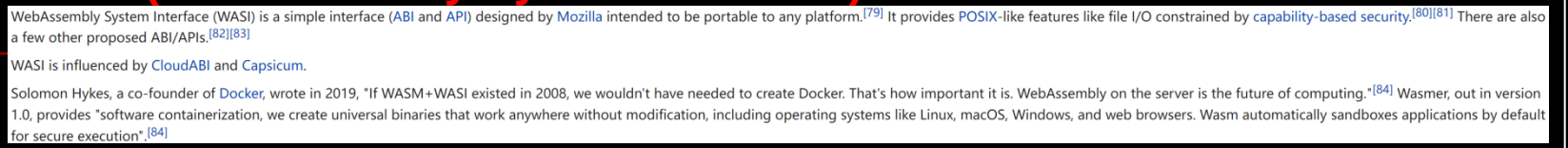
The first of these ensures the proper memory isolation of the sandbox, while the latter two provide control-flow integrity of the sandbox.

Source: https://github.com/gwsystems/aWsm/blob/master/doc/design.md

1.2.3 Beyond the browser

1.2.3.1 WASI

- **WASI (WebAssembly System Interface)**



WebAssembly System Interface (WASI) is a simple interface (ABI and API) designed by Mozilla intended to be portable to any platform.[79] It provides POSIX-like features like file I/O constrained by capability-based security.[80][81] There are also a few other proposed ABI/APIs.[82][83]

WASI is influenced by CloudABI and Capsicum.

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]

- **https://wasi.dev/**
- **https://github.com/WebAssembly/WASI**
- **https://webassembly.org/docs/non-web/**
- **https://github.com/bytecodealliance/wasmtime/blob/main/docs/WASI-documents.md**
- **https://hacks.mozilla.org/2019/03/standardizing-wasi-a-webassembly-system-interface/**
- **https://training.linuxfoundation.org/announcements/wasi-bringing-webassembly-way-beyond-browsers**
- **https://www.zdnet.com/article/microsoft-google-back-bytecode-alliance-to-move-webassembly-beyond-the-browser/**
- **…**

1.2.4 Wasm 2.0

- https://www.w3.org/blog/news/archives/9509

The WebAssembly Working Group has published three First Public Working Drafts:

- WebAssembly Core Specification – Version 2.0 describes version 2.0 of the core WebAssembly standard, a safe, portable, low-level code format designed for efficient execution and compact representation.
- WebAssembly JavaScript Interface – Version 2.0 provides an explicit JavaScript API for interacting with WebAssembly.
- WebAssembly Web API – Version 2.0 describes the integration of WebAssembly with the broader web platform.

The WebAssembly Working Group maintains a list of proposals finished since the last Recommendation and monitors the progress of "in-flight" proposals.

- https://github.com/WebAssembly/proposals
- …

Roadmap

- https://webassembly.org/roadmap/

	Your browser	Chrome	Firefox	Safari	Wasmtime	Wasmer	Node.js	Deno	wasm2c
Standardized features									
JS BigInt to Wasm i64 integration	✓	85	78	14.1[f]	N/A	N/A	✓	✓	N/A
Bulk memory operations	✓	75	79	15	✓	✓	✓	✓	✓
Multi-value	✓	85	78	✓	✓	✓	✓	✓	✓
Import & export of mutable globals	✓	74	61	✓	✓	✓	✓	✓	✓
Reference types	✓	96	79	15	✓	✓	⚑[m]	✓	✓
Non-trapping float-to-int conversions	✓	75	64	15	✓	✓	✓	✓	✓
Sign-extension operations	✓	74	62	14.1[f]	✓	✓	✓	✓	✓
Fixed-width SIMD	✓	91	89	✗	✓	✓	✓	✓	✗
In-progress proposals									
Exception handling	✓	95	100	15.2	✗	✗	⚑[k]	✓	⚑[o]
Extended constant expressions	✗	⚑[a]	⚑[e]	✗	✗	✗	⚑[l]	✗	✗
Memory64	✗	⚑[b]	⚑[e]	✗	⚑[g]	✗	✗	✗	✗
Multiple memories	?	✗	✗	✗	⚑[h]	✗	✗	✗	⚑[h]
Module Linking	?	✗	✗	✗	⚑[i]	✗	✗	✗	✗
Relaxed SIMD	?	⚑[c]	⚑[e]	✗	✗	✗	✗	✗	✗
Tail calls	✗	⚑[d]	✗	✗	✗	✗	⚑[n]	✗	✗
Threads and atomics	✓	74	79	14.1[f]	✗	⚑[j]	✓	✓	✗
Type reflection	?	✗	⚑[e]	✗	✗	✗	✗	✗	✗

1.3 .Net

1.3.1 Overview

- https://en.wikipedia.org/wiki/.NET
- https://dotnet.microsoft.com/
- https://en.wikipedia.org/wiki/Windows_Runtime
- https://en.wikipedia.org/wiki/.NET_Framework
- https://github.com/quozd/awesome-dotnet
- https://www.infoq.com/articles/dotnet-trends-2022/

History

Version	Release date	Released with	Latest update	Latest update date	Support ends[21]
.NET Core 1.0	2016-06-27[22]	Visual Studio 2015 Update 3	1.0.16	May 14, 2019	June 27, 2019
.NET Core 1.1	2016-11-16[23]	Visual Studio 2017 Version 15.0	1.1.13	May 14, 2019	June 27, 2019
.NET Core 2.0	2017-08-14[24]	Visual Studio 2017 Version 15.3	2.0.9	July 10, 2018	October 1, 2018
.NET Core 2.1	2018-05-30[25]	Visual Studio 2017 Version 15.7	2.1.30 (LTS)	August 19, 2021	August 21, 2021
.NET Core 2.2	2018-12-04[26]	Visual Studio 2019 Version 16.0	2.2.8	November 19, 2019	December 23, 2019
.NET Core 3.0	2019-09-23[27]	Visual Studio 2019 Version 16.3	3.0.3	February 18, 2020	March 3, 2020
.NET Core 3.1	2019-12-03[28]	Visual Studio 2019 Version 16.4	3.1.30 (LTS)	October 11, 2022	December 13, 2022
.NET 5	2020-11-10[29]	Visual Studio 2019 Version 16.8	5.0.17	May 10, 2022	May 10, 2022
.NET 6	2021-11-08[30]	Visual Studio 2022 Version 17.0	6.0.11 (LTS)	November 8, 2022	November 12, 2024
.NET 7	2022-11-08	Visual Studio 2022 Version 17.4	7.0.0	November 8, 2022	May 14, 2024
.NET 8	2023-11 (projected)		(will be LTS)		November 2026 (projected)

Old version | Older version, still maintained | **Latest version** | Future release

Source: https://en.wikipedia.org/wiki/.NET

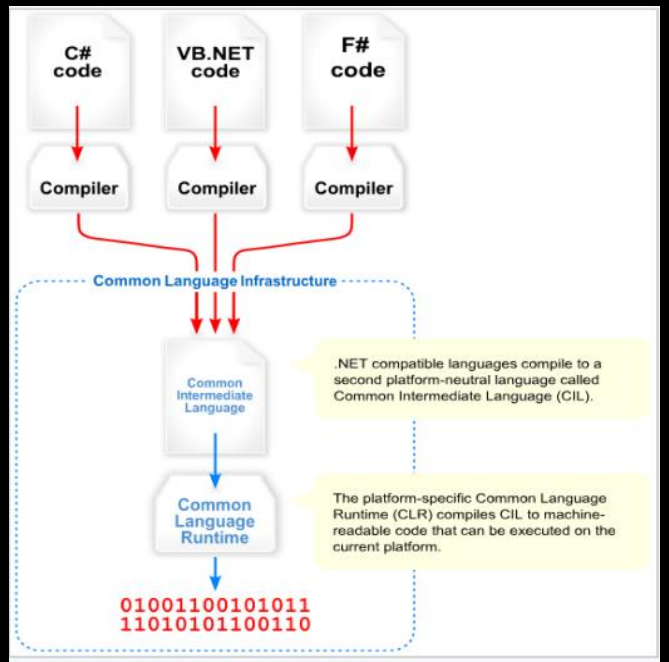
- https://devblogs.microsoft.com/dotnet/happy-20th-anniversary-net/

1.3.2 Runtime

CLI

- https://en.wikipedia.org/wiki/Common_Language_Infrastructure

The **Common Language Infrastructure** (**CLI**) is an open specification and technical standard originally developed by Microsoft and standardized by ISO (**ISO/IEC 23271**) and Ecma International (**ECMA 335**)[1][2] that describes executable code and a runtime environment that allows multiple high-level languages to be used on different computer platforms without being rewritten for specific architectures. This implies it is platform agnostic. The .NET Framework, .NET and Mono are implementations of the CLI. The metadata format is also used to specify the API definitions exposed by the Windows Runtime.[3][4]

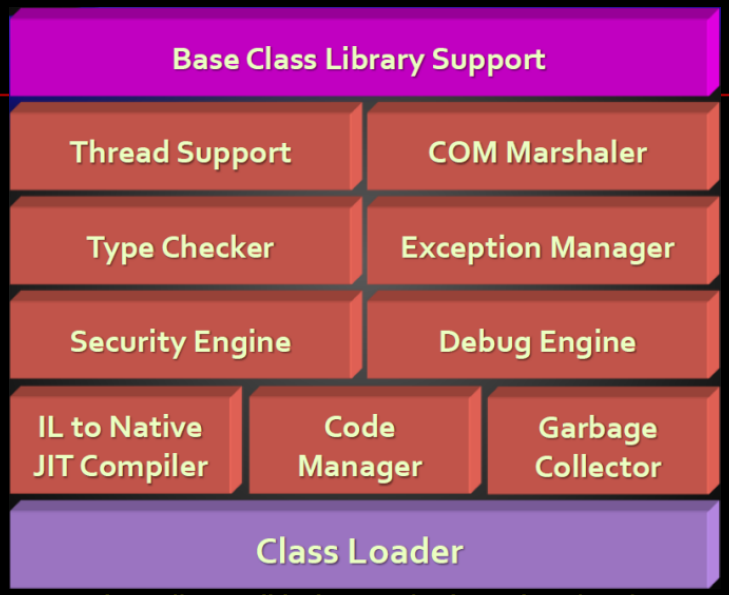


Visual overview of the Common Language Infrastructure (CLI)

- https://en.wikipedia.org/wiki/Common_Intermediate_Language

CLR

- https://en.wikipedia.org/wiki/Common_Language_Runtime
- Architecture



Source: https://www.slideshare.net/MohamadKrm/net-framework-250644809

- https://docs.microsoft.com/en-us/dotnet/standard/clr

Src

- https://github.com/dotnet/runtime

```
[mydev@fedora runtime-main]$ tree -L 1
.
├── build.cmd
├── Build.proj
├── build.sh
├── CODE-OF-CONDUCT.md
├── CONTRIBUTING.md
├── Directory.Build.props
├── Directory.Build.targets
├── Directory.Solution.props
├── docs
├── dotnet.cmd
├── dotnet.sh
├── eng
├── global.json
├── LICENSE.TXT
├── NuGet.config
├── PATENTS.TXT
├── README.md
├── SECURITY.md
├── src
└── THIRD-PARTY-NOTICES.TXT
```

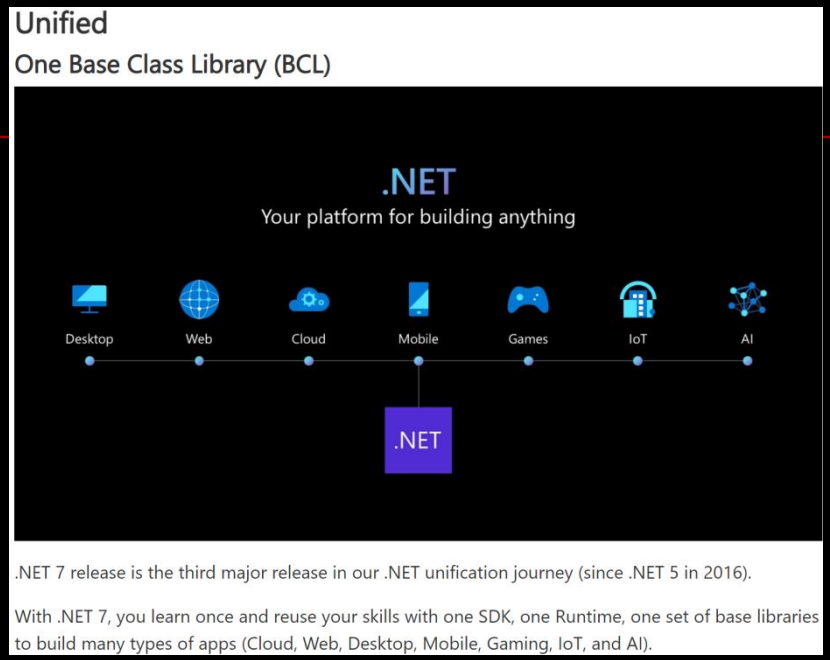
```
[mydev@fedora runtime-main]$ tree -L 1 src
src
├── coreclr
├── installer
├── libraries
├── mono
├── native
├── samples
├── tasks
├── tests
├── tools
└── workloads
```

```
[mydev@fedora runtime-main]$ tree -L 1 src/coreclr
src/coreclr
├── binder
├── _build-commons.sh
├── build-runtime.cmd
├── build-runtime.sh
├── classlibnative
├── clrdefinitions.cmake
├── clr.featuredefines.props
├── clrfeatures.cmake
├── CMakeLists.txt
├── CMakeSettings.json
├── components.cmake
├── cpp.hint
├── crosscomponents.cmake
├── crossgen-corelib.proj
├── debug
├── Directory.Build.props
├── Directory.Build.targets
├── dlls
├── enablesanitizers.sh
├── gc
├── gcdump
├── gcinfo
├── generateredefinesfile.sh
├── hosts
├── ilasm
├── ildasm
├── inc
├── interop
├── jit
├── md
├── minipal
├── nativeaot
├── nativeresources
├── pal
├── palrt
├── pgosupport.cmake
├── run-cppcheck.sh
├── runtime-prereqs.proj
├── runtime.proj
├── scripts
├── System.Private.CoreLib
├── tools
├── unwinder
├── utilcode
└── vm
```

- https://github.com/dotnet/llvm-project

1.3.3 .Net 7

- https://devblogs.microsoft.com/dotnet/announcing-dotnet-7/



- https://learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-7
- https://learn.microsoft.com/en-us/dotnet/core/compatibility/7.0?toc=%2Fdotnet%2Ffundamentals%2Ftoc.json&bc=%2Fdotnet%2Fbreadcrumb%2Ftoc.json

C# 11

- https://devblogs.microsoft.com/dotnet/welcome-to-csharp-11/
- …

Native AOT

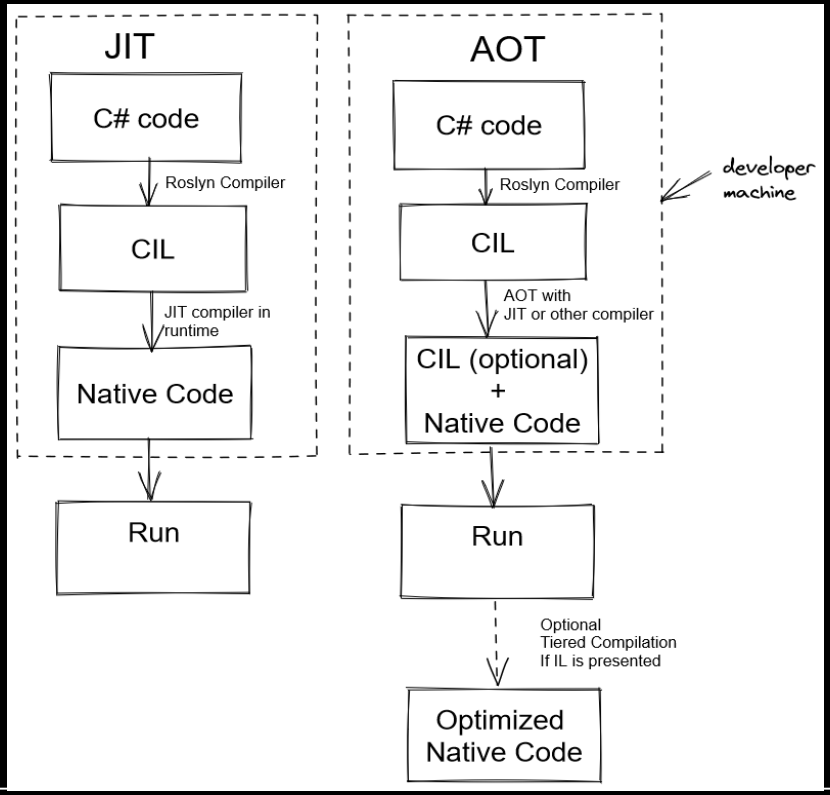
- https://flerka.github.io/personal-blog/2022-06-21-ahead-of-time-compilation/

What is AOT

When we talk about ahead-of-time (AOT) compilation, we usually mean tools that allow us to generate native code in build-time instead of runtime (JIT compilation). AOT binaries will be larger because they contain not only native code but an IL code too.

AOT vs JIT compilation

JIT executes on the end-user/server machine, and AOT usually (ngen.exe is an exception, for example) runs on the developer machine after Roslyn compilation.



Benefits

- it helps to improve startup performance.
- it can produces libraries that can be consumed by programs written in programming languages that don't use CLR.
- it can be used on the platform where JIT, for some reason, is not allowed. For example, Xamarin.iOS uses ahead-of-time compilation because of security restrictions iOS has. Apple doesn't allow the execution of dynamically generated code on iOS.

Tools and approaches

- ReadyToRun (R2R) - current mainstream approach for AOT compilation in dotnet. It describes an assembly format that contains IL + native code. In .NET 6.0 its performance was improved with Composite R2R Compilation. This feature allows cross-assembly optimization, such as the inlining method from another assembly. ReadyToRun supports cross-compilation.
- mono AOT - supports several modes: full and partial AOT. Full AOT for systems where JIT can't be run.
- crossgen tool - later it was replaced with crossgen2.
- crossgen2 tool - can generate assemblies in ReadyToRun format. It is the second version of the crossgen tool that the .NET team rewrote in C#. Crossgen2 has added features, such as version bubbles. The crossgen2 architecture uses a graph that allows performing different analysis and optimizations. There is great article that discusses crossgen2.
- ngen.exe - for .Net framework. It runs on the target machine and produces executables that are not self-contained and requires an installed runtime. Can cause some performance issues on target PC.
- CoreRT - .NET Core runtime optimized for ahead-of-time compilation. Right now, this project has been replaced by Native AOT. It could use not only JIT compiler but experimental projects, like IL to CPP. It generates one file containing the app itself, its dependencies, and CoreRT runtime. If you want to know more, you can read a great post from Matt Warren that has everything nicely written and illustrated.
- .NET Native - as far as I know, it's for UWP only.
- IL2CPP - it compiles IL assemblies to native code, not open source, and for Unity only.
- IL2CPU - it's used in Cosmos project.

- **Deploy Lambdas and Compare their execution**

Compilation type	First run in ms (init duration is not included)	Second run in ms	Third run in ms
Native AOT .NET 7 on provided.al2	39.49	0.83	0.80
R2R .NET 6 on dotnet6	273	1.51	9.09
R2R .NET 7 on provided.al2	457.28	37.98	1.38
Regular .NET 6 on dotnet6	353.86	36.30	1.00
Regular .NET 7 on provided.al2	446.86	9.23	1.21

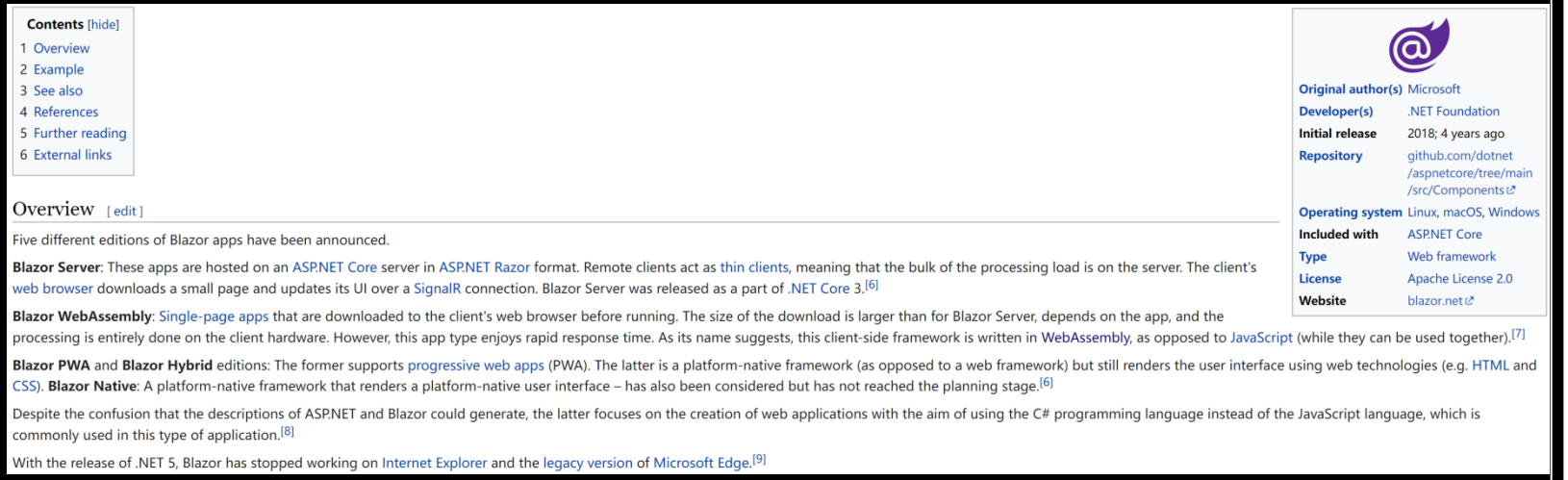
As you can see, Native AOT shows better numbers for the warm and cold start. ReadyToRun in dotnet runtime also shows better numbers for the cold start than regular projects and self-contained R2R.

- https://visualstudiomagazine.com/articles/2022/04/15/net-7-preview-3.aspx
- https://devblogs.microsoft.com/dotnet/announcing-dotnet-7-preview-3/#faster-lighter-apps-with-native-aot
- https://learn.microsoft.com/en-us/dotnet/core/deploying/native-aot/
- https://nodogmablog.bryanhogan.net/2022/11/lambda-cold-starts-net-7-native-aot-vs-net-6-managed-runtime/
- …

1.3.4 Wasm on .Net

Blazor

- https://en.wikipedia.org/wiki/Blazor
  A free and open-source web framework that enables developers to create web apps using C# and HTML.



Contents [hide]
1 Overview
2 Example
3 See also
4 References
5 Further reading
6 External links

Overview [edit]

Five different editions of Blazor apps have been announced.

Blazor Server: These apps are hosted on an ASP.NET Core server in ASP.NET Razor format. Remote clients act as thin clients, meaning that the bulk of the processing load is on the server. The client's web browser downloads a small page and updates its UI over a SignalR connection. Blazor Server was released as a part of .NET Core 3.[6]

Blazor WebAssembly: Single-page apps that are downloaded to the client's web browser before running. The size of the download is larger than for Blazor Server, depends on the app, and the processing is entirely done on the client hardware. However, this app type enjoys rapid response time. As its name suggests, this client-side framework is written in WebAssembly, as opposed to JavaScript (while they can be used together).[7]

Blazor PWA and **Blazor Hybrid** editions: The former supports progressive web apps (PWA). The latter is a platform-native framework (as opposed to a web framework) but still renders the user interface using web technologies (e.g. HTML and CSS). **Blazor Native**: A platform-native framework that renders a platform-native user interface – has also been considered but has not reached the planning stage.[6]

Despite the confusion that the descriptions of ASP.NET and Blazor could generate, the latter focuses on the creation of web applications with the aim of using the C# programming language instead of the JavaScript language, which is commonly used in this type of application.[8]

With the release of .NET 5, Blazor has stopped working on Internet Explorer and the legacy version of Microsoft Edge.[9]

Original author(s)	Microsoft
Developer(s)	.NET Foundation
Initial release	2018; 4 years ago
Repository	github.com/dotnet/aspnetcore/tree/main/src/Components
Operating system	Linux, macOS, Windows
Included with	ASP.NET Core
Type	Web framework
License	Apache License 2.0
Website	blazor.net

https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor
https://github.com/dotnet/aspnetcore
https://github.com/AdrienTorris/awesome-blazor

WASI SDK for .NET (Experimental)

- https://github.com/dotnet/dotnet-wasi-sdk
- https://github.com/SteveSandersonMS/dotnet-wasi-sdk

`Wasi.Sdk` is an experimental package that can build .NET Core projects (including whole ASP.NET Core applications) into standalone WASI-compliant `.wasm` files. These can then be run in standard WASI environments or custom WASI-like hosts.


```
[submodule "modules/runtime"]
        path = modules/runtime
        url = https://github.com/dotnet/runtime.git
```

What's in this repo

- `Wasi.Sdk` - a package that causes your build to produce a WASI-compliant `.wasm` file. This works by:
  - Downloading the WASI SDK, if you don't already have it
  - When your regular .NET build is done, it takes the resulting assemblies, plus the .NET runtime precompiled to WebAssembly, and uses WASI SDK to bundle them into a single `.wasm` file. You can optionally include other native sources such as `.c` files in the compilation.
- `Wasi.AspNetCore.BundledFiles` - provides `UseBundledStaticFiles`, and alternative to `UseStaticFiles`, that serves static files bundled into your `.wasm` file. This allows you to have single-file deployment even if you have files under `wwwroot` or elsewhere.
- `Wasi.AspNetCore.Server.Native` - a way of running ASP.NET Core on WASI's TCP-level standard networking APIs (e.g., `sock_accept`). These standards are quite recent and are currently only supported in Wasmtime, not other WASI hosts.

... and more

```
[mydev@fedora dotnet-wasi-sdk]$ tree -L 2
.
├── Directory.Build.props
├── dotnet-wasi-sdk.sln
├── modules
│   └── runtime
├── README.md
├── samples
│   ├── AspNetCoreOnCustomHost
│   ├── AspNetCoreOnNativeWasi
│   ├── atmo
│   ├── ConsoleApp
│   ├── Directory.Build.props
│   └── webserver-wasi-host
└── src
    ├── Wasi.AspNetCore.BundledFiles
    ├── Wasi.AspNetCore.Server
    ├── Wasi.AspNetCore.Server.Atmo
    ├── Wasi.AspNetCore.Server.CustomHost
    ├── Wasi.AspNetCore.Server.Native
    └── Wasi.Sdk
```

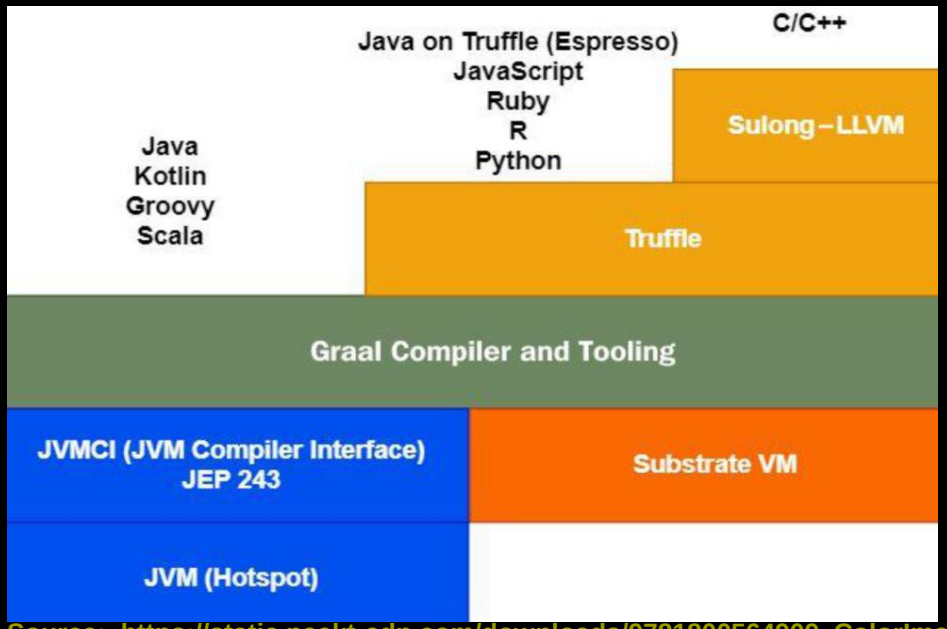
...

```
[mydev@fedora dotnet-wasi-sdk]$ tree src
src
├── Wasi.AspNetCore.BundledFiles
│   ├── build
│   │   ├── Wasi.AspNetCore.BundledFiles.props
│   │   └── Wasi.AspNetCore.BundledFiles.targets
│   ├── native
│   │   └── interop.c
│   ├── Wasi.AspNetCore.BundledFiles.csproj
│   ├── WasiBundledFileProvider.cs
│   └── WasiNativeWebApplicationExtensions.cs
├── Wasi.AspNetCore.Server
│   ├── Wasi.AspNetCore.Server.csproj
│   ├── WasiLoggingProvider.cs
│   ├── WasiServer.cs
│   └── WasiServerRequestContext.cs
├── Wasi.AspNetCore.Server.Atmo
│   ├── AtmoLogger.cs
│   ├── AtmoRequestContext.cs
│   ├── AtmoServer.cs
│   ├── AtmoWebApplicationBuilderExtensions.cs
│   ├── build
│   │   ├── Wasi.AspNetCore.Server.Atmo.props
│   │   └── Wasi.AspNetCore.Server.Atmo.targets
│   ├── Interop.cs
│   ├── native
│   │   └── host_interop.c
│   ├── Services
│   │   ├── Cache.cs
│   │   └── IdentAccessor.cs
│   └── Wasi.AspNetCore.Server.Atmo.csproj
├── Wasi.AspNetCore.Server.CustomHost
│   ├── build
│   │   ├── Wasi.AspNetCore.Server.CustomHost.props
│   │   └── Wasi.AspNetCore.Server.CustomHost.targets
│   ├── CustomHostWebApplicationBuilderExtensions.cs
│   ├── Interop.cs
│   ├── native
│   │   └── server_interop.c
│   ├── Wasi.AspNetCore.Server.CustomHost.csproj
│   ├── WasiCustomHostRequestContext.cs
│   └── WasiCustomHostServer.cs
├── Wasi.AspNetCore.Server.Native
│   ├── build
│   │   ├── Wasi.AspNetCore.Server.Native.props
│   │   └── Wasi.AspNetCore.Server.Native.targets
│   ├── DuplexPipe.cs
│   ├── Interop.cs
│   ├── native
│   │   ├── dotnet_method.h
│   │   ├── interop.c
│   │   └── tcp_listener_loop.c
│   ├── Wasi.AspNetCore.Server.Native.csproj
│   ├── WasiConnectionContext.cs
│   ├── WasiConnectionListener.cs
│   ├── WasiConnectionListenerFactory.cs
│   ├── WasiLoggingProvider.cs
│   ├── WasiNativeServer.cs
│   └── WasiNativeWebApplicationBuilderExtensions.cs
└── Wasi.Sdk
    ├── build
    │   ├── Wasi.Sdk.props
    │   └── Wasi.Sdk.targets
    ├── native
    │   └── main.c
    ├── Tasks
    │   ├── EmitWasmBundleObjectFile.cs
    │   └── WasmResolveAssemblyDependencies.cs
    └── Wasi.Sdk.csproj
```

1.4 GraalVM

1.4.1 Overview

- https://en.wikipedia.org/wiki/GraalVM
- https://www.graalvm.org/
-

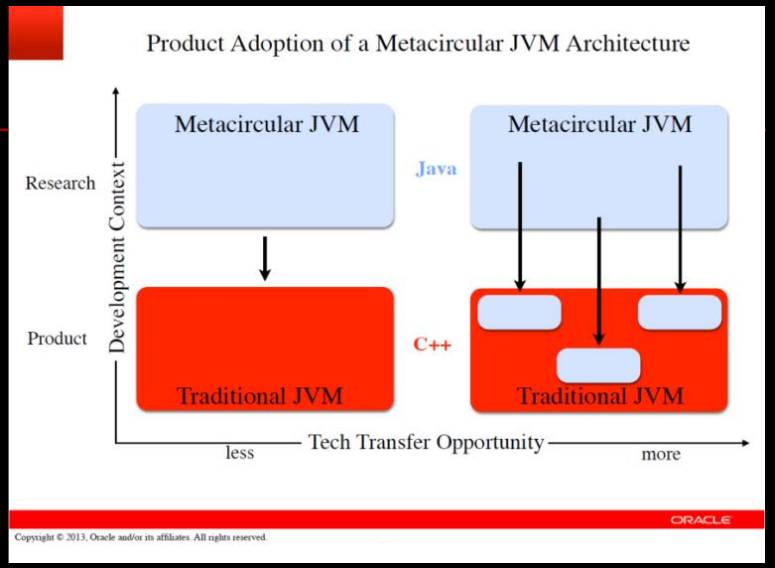


Source: https://static.packt-cdn.com/downloads/9781800564909_ColorImages.pdf

- A Universal High-Performance Polyglot VM
- A meta-runtime for Language-Level Virtualization
- Base on OpenJDK 8, 11, 16, 17 and 19 with JVMCI support.
- https://github.com/graalvm
- https://github.com/oracle/graal

Meta-Circular

- MaxineVM → GraalVM

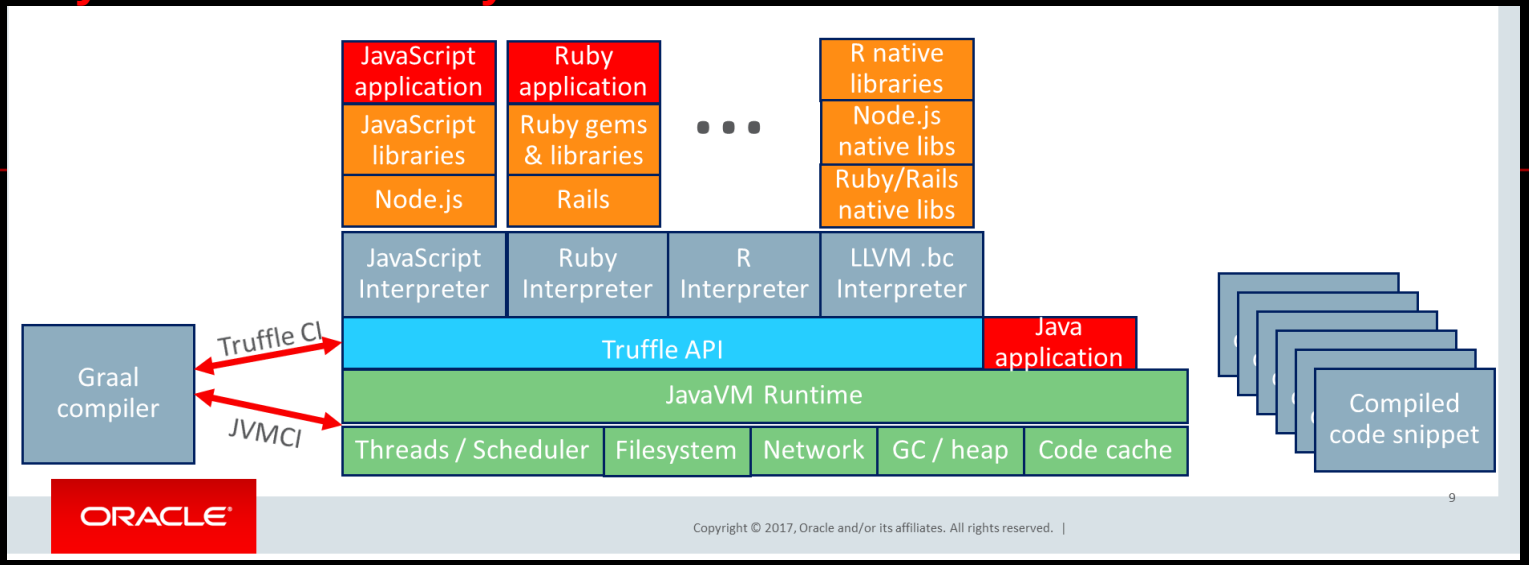


Source: https://chrisseaton.com/truffleruby/jokerconf17/

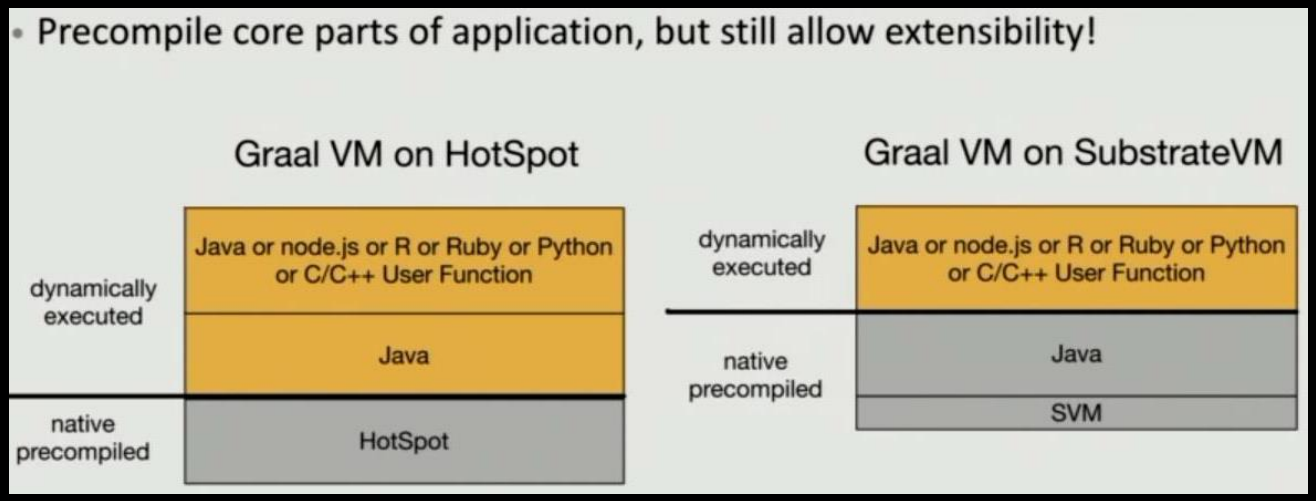
- https://en.wikipedia.org/wiki/Maxine_Virtual_Machine
- https://github.com/beehive-lab/Maxine-VM
- https://maxine-vm.readthedocs.io/en/stable/
- …

Architecture & design

- A hybrid of static & dynamic runtimes

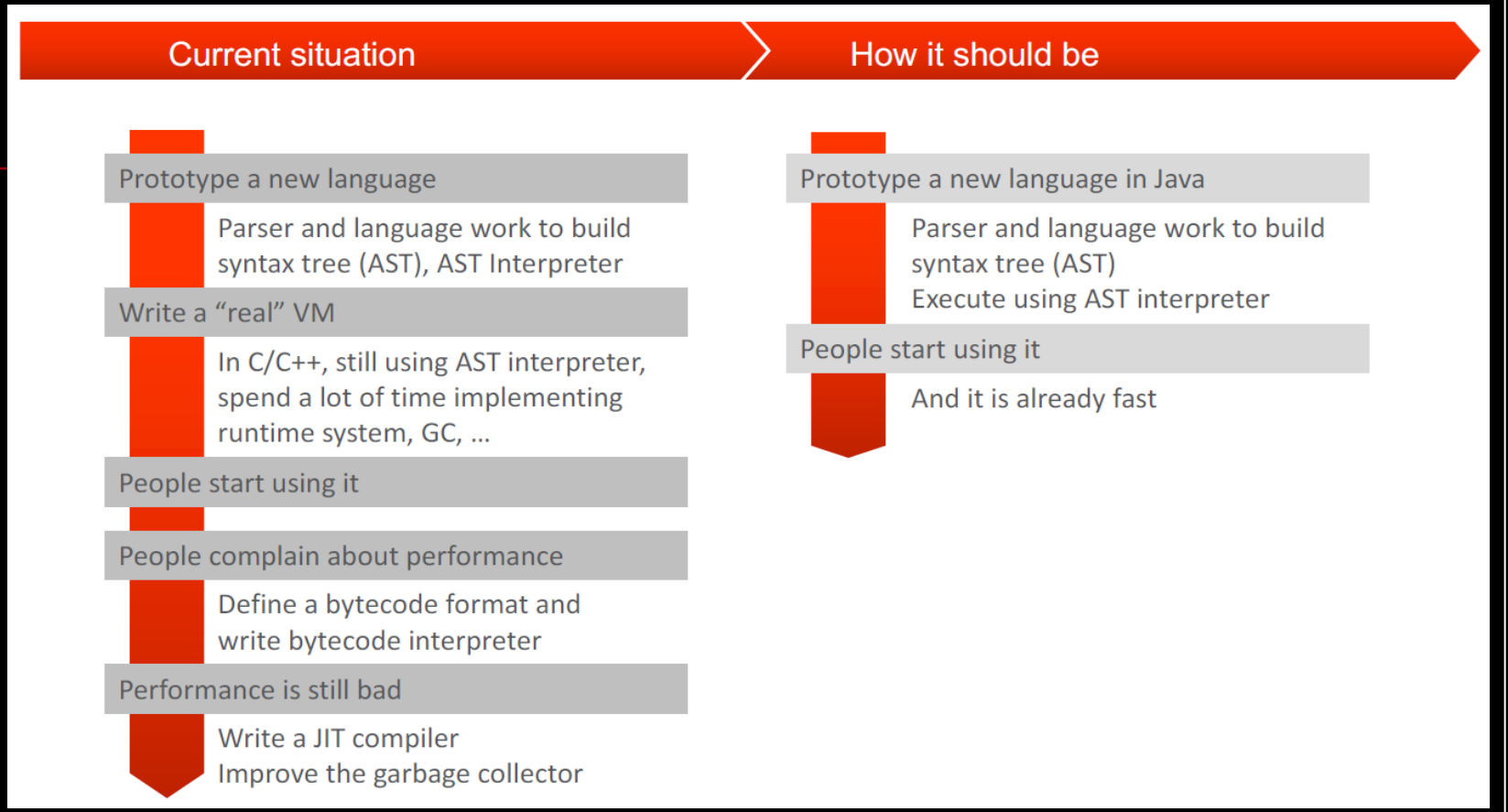


Source: https://ics.psu.edu/wp-content/uploads/2017/02/GraalVM-PSU.pptx



Source: "Adopting Java for the Serverless world", Vadym Kazulkin, JUG London 2020.

Implement a new language runtime



Source: "Turning the JVM into a Polyglot VM with Graal", Chris Seaton, Oracle Labs.

- **https://www.graalvm.org/22.0/graalvm-as-a-platform/language-implementation-framework/Languages/**

1.4.2 Current release and roadmap

- https://medium.com/graalvm/announcing-the-graalvm-community-roadmap-b8d77201b497

GraalVM 22.2

- https://medium.com/graalvm/graalvm-22-2-smaller-jdk-size-improved-memory-usage-better-library-support-and-more-cb34b5b68ec0

  Smaller JDK size, improved memory usage, better library support, and more!

GraalVM version and platform	JDK size in 22.1	JDK size in 22.2	Difference
GraalVM Community / Java 17 / Linux AMD64	431 MB	251 MB	-42%
GraalVM Community / Java 17 / Darwin AMD64	425 MB	247 MB	-42%
GraalVM Enterprise / Java 17 / Darwin AMD64	495 MB	271 MB	-45%

...

JDK 19

- https://openjdk.java.net/projects/jdk/19/

  Features

405: Record Patterns (Preview)
422: Linux/RISC-V Port
424: Foreign Function & Memory API (Preview)
425: Virtual Threads (Preview)
426: Vector API (Fourth Incubator)
427: Pattern Matching for switch (Third Preview)
428: Structured Concurrency (Incubator)

- https://jdk.java.net/19/release-notes

1.4.3 Languages

- https://github.com/oracle/graal/blob/master/truffle/docs/Languages.md

Language Implementations

This page is intended to keep track of the growing number of language implementations and experiments on top of Truffle. The following language implementations exist already:

- Espresso, a meta-circular Java bytecode interpreter. *
- FastR, an implementation of GNU R. *
- Graal.js, an ECMAScript 2020 compliant JavaScript implementation. *
- GraalPy, an early-stage implementation of Python. *
- grCUDA, a polyglot CUDA integration.
- SimpleLanguage, a toy language implementation to demonstrate Truffle features.
- SOMns, a Newspeak implementation for Concurrency Research.
- Sulong, an LLVM bitcode interpreter. *
- TRegex, a generic regular expression engine (internal, for use by other languages only). *
- TruffleRuby, an implementation of Ruby. *
- TruffleSOM, a SOM Smalltalk implementation.
- TruffleSqueak, a Squeak/Smalltalk VM implementation and polyglot programming environment.
- Yona, the reference implementation of a minimalistic, strongly and dynamically-typed, parallel and non-blocking, polyglot, strict, functional programming language.
- Enso, an open source, visual language for data science that lets you design, prototype and develop any application by connecting visual elements together.

* Shipped as part of GraalVM.

Experiments

- BACIL, .NET CIL interpreter.
- bf, an experimental Brainfuck programming language implementation.
- brainfuck-jvm, another Brainfuck language implementation.
- Cover, a Safe Subset of C++.
- DynSem, a DSL for declarative specification of dynamic semantics of languages.
- Heap Language, a tutorial showing the embedding of Truffle languages via interoperability.
- hextruffe, an implementation of Hex.
- LuaTruffle, an implementation of the Lua language.
- Mozart-Graal, an implementation of the Oz programming language.
- Mumbler, an experimental Lisp programming language.
- PorcE, an Orc language implementation.
- ProloGraal a Prolog language implementation supporting interoperability.
- PureScript, a small, strongly-typed programming language.
- Reactive Ruby, TruffleRuby meets Reactive Programming.
- shen-truffle, a port of the Shen programming language.
- TruffleMATE, a Smalltalk with a completely reified runtime system.
- TrufflePascal, a Pascal interpreter.
- ZipPy, a Python implementation.

...

1.4.4 Wasm

1.4.4.1 GraalWasm

- https://github.com/oracle/graal/tree/master/wasm

GraalWasm is a WebAssembly engine implemented in GraalVM. It can interpret and compile WebAssembly programs in the binary format, or be embedded into other programs.

...

Embedding GraalWasm inside other programs

GraalWasm can be accessed programmatically with the Polyglot API, which allows embedding GraalWasm into user programs.

Here is a simple example of how to run a WebAssembly program using GraalWasm from a Java application:

```
import org.graalvm.polyglot.*;
import org.graalvm.polyglot.io.ByteSequence;

byte[] binary = readBytes("example.wasm"); // You need to load the .wasm contents into a byte array.
Context.Builder contextBuilder = Context.newBuilder("wasm");
Source.Builder sourceBuilder = Source.newBuilder("wasm", ByteSequence.create(binary), "example");
Source source = sourceBuilder.build();
Context context = contextBuilder.build();

context.eval(source);

Value mainFunction = context.getBindings("wasm").getMember("example").getMember("_main");
mainFunction.execute();
```

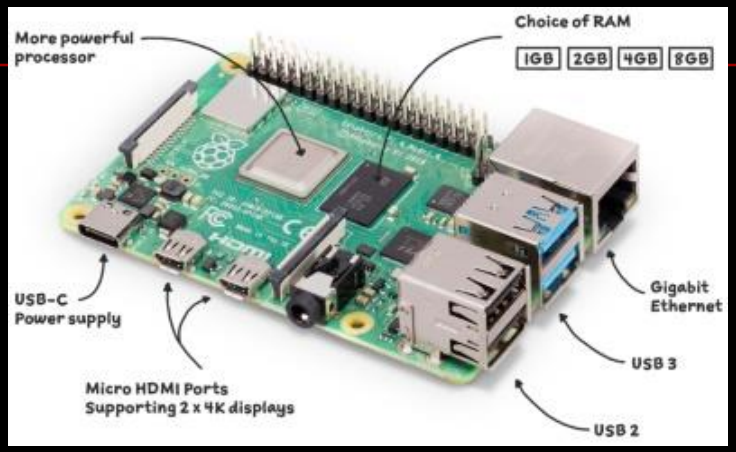
```
[mydev@fedora /]$ which wasm
/opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/v22.3.0-dev-20221120/Java19/bin/wasm
[mydev@fedora /]$
[mydev@fedora /]$ file /opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/v22.3.0-dev-20221120/Java19/bin/wasm
/opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/v22.3.0-dev-20221120/Java19/bin/wasm: symbolic link to ../languages/wasm/bin/wasm
[mydev@fedora /]$
[mydev@fedora /]$ file /opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/v22.3.0-dev-20221120/Java19/languages/wasm/bin/wasm
/opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/v22.3.0-dev-20221120/Java19/languages/wasm/bin/wasm: ELF 64-bit LSB executable, ARM aarch64, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux
-aarch64.so.1, for GNU/Linux 3.7.0, BuildID[sha1]=5ff6794be91582b2e2ec1f287eddc5bbc18186bb, with debug_info, not stripped
[mydev@fedora /]$
```

- https://www.graalvm.org/22.3/reference-manual/wasm/
- https://subscription.packtpub.com/book/programming/9781800564909/12/ch12lvl1sec86/understanding-graalwasm-the-wasm-truffle-interpreter
- ...

2) Testbed

2.1 Raspberry Pi 4(8GB LPDDR4) with Fedora 37

- HW env



SW env

```
[mydev@fedora /]$ uname -a
Linux fedora 6.0.17-300.fc37.aarch64 #1 SMP PREEMPT_DYNAMIC Wed Jan 4 15:36:28 UTC 2023 aarch64 aarch64 aarch64 GNU/Linux
[mydev@fedora /]$
[mydev@fedora /]$ free -m
               total        used        free      shared  buff/cache   available
Mem:            7826         488        4387          16        2950        7059
Swap:           7825           0        7825
[mydev@fedora /]$ _
```

```
[mydev@fedora /]$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/aarch64-redhat-linux/12/lto-wrapper
Target: aarch64-redhat-linux
Configured with: ../configure --enable-bootstrap --enable-languages=c,c++,fortran,objc,obj-c++,ada,go,d,lto --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --with-bugurl=http://bugzilla.redhat.com/bugzilla --enable-shared --enable-threads=posix --enable-checking=release --enable-multilib --with-system-zlib --enable-__cxa_atexit --disable-libunwind-exceptions --enable-gnu-unique-object --enable-linker-build-id --with-gcc-major-version-only --enable-libstdcxx-backtrace --with-linker-hash-style=gnu --enable-plugin --enable-initfini-array --with-isl=/builddir/build/BUILD/gcc-12.2.1-20221121/obj-aarch64-redhat-linux/isl-install --enable-gnu-indirect-function --build=aarch64-redhat-linux --with-build-config=bootstrap-lto --enable-link-serialization=1
Thread model: posix
Supported LTO compression algorithms: zlib zstd
gcc version 12.2.1 20221121 (Red Hat 12.2.1-4) (GCC)
[mydev@fedora /]$
[mydev@fedora /]$ clang -v
clang version 15.0.6 (Fedora 15.0.6-2.fc37)
Target: aarch64-redhat-linux-gnu
Thread model: posix
InstalledDir: /usr/bin
Found candidate GCC installation: /usr/bin/../lib/gcc/aarch64-redhat-linux/12
Selected GCC installation: /usr/bin/../lib/gcc/aarch64-redhat-linux/12
Candidate multilib: .;@m64
Selected multilib: .;@m64
[mydev@fedora /]$
```

```
[mydev@fedora /]$ python -V
Python 3.11.1
[mydev@fedora /]$
```

```
[mydev@fedora /]$ graalpy -V
GraalPy 3.10.8 (GraalVM CE Native 23.0.0-dev)
[mydev@fedora /]$
```

```
[mydev@fedora /]$ java --version
openjdk 19.0.1 2022-10-18
OpenJDK Runtime Environment GraalVM CE 23.0.0-dev (build 19.0.1+10-jvmci-23.0-b04)
OpenJDK 64-Bit Server VM GraalVM CE 23.0.0-dev (build 19.0.1+10-jvmci-23.0-b04, mixed mode, sharing)
[mydev@fedora /]$
[mydev@fedora /]$ gu list
ComponentId              Version             Component name                Stability
------------------------------------------------------------------------------------------
------
graalvm                  23.0.0-dev          GraalVM Core                  Experimental
js                       23.0.0-dev          Graal.js                      Experimental
llvm                     23.0.0-dev          LLVM Runtime Core             Experimental
llvm-toolchain           23.0.0-dev          LLVM.org toolchain            Experimental
native-image             23.0.0-dev          Native Image                  Experimental
native-image-llvm-backend23.0.0-dev          Native Image LLVM Backend     Experimental
nodejs                   23.0.0-dev          Graal.nodejs                  Experimental
python                   23.0.0-dev          GraalVM Python                Experimental
visualvm                 23.0.0-dev          VisualVM                      Experimental
wasm                     23.0.0-dev          GraalWasm                     Experimental
[mydev@fedora /]$
```

```
[mydev@fedora /]$ dotnet --version
7.0.100
[mydev@fedora /]$ mono --version
Mono JIT compiler version 6.12.0.182 (tarball Thu Jul 21 23:55:50 UTC 2022)
Copyright (C) 2002-2014 Novell, Inc, Xamarin Inc and Contributors. www.mono-project.com
        TLS:           __thread
        SIGSEGV:       normal
        Notifications: epoll
        Architecture:  arm64
        Disabled:      none
        Misc:          softdebug
        Interpreter:   yes
        LLVM:          supported, not enabled.
        Suspend:       preemptive
        GC:            sgen (concurrent by default)
[mydev@fedora /]$ _
```

```
[mydev@fedora /]$ rustup show
Default host: aarch64-unknown-linux-gnu
rustup home:  /home/mydev/.rustup

installed targets for active toolchain
--------------------------------------

aarch64-unknown-linux-gnu
wasm32-unknown-unknown

active toolchain
----------------

stable-aarch64-unknown-linux-gnu (default)
rustc 1.66.0 (69f9c33d7 2022-12-12)

[mydev@fedora /]$
```

2.1.1 Fedora

- A Linux distribution developed by the community-supported Fedora Project which is sponsored primarily by Red Hat.
- https://en.wikipedia.org/wiki/Fedora_(operating_system)
- https://getfedora.org/
- https://alt.fedoraproject.org/alt/
- https://spins.fedoraproject.org/
- https://fedoraproject.org/wiki/Architectures/ARM
- https://fedoramagazine.org/
- https://silverblue.fedoraproject.org/
- https://fedoraproject.org/wiki/Changes/RaspberryPi4
- Developer friendly!

2.1.1.1 Fedora 37

- https://fedoraproject.org/wiki/Releases/37/ChangeSet
- https://www.phoronix.com/news/Fedora-37-Released

Raspberry Pi 4

- **Fedora 37 To Offer Official Support On Raspberry Pi 4 Devices**

A month ago there was the Fedora 37 change proposal for Fedora to officially support the Raspberry Pi 4, including its accelerated Broadcom graphics and to better advertise Fedora for the Raspberry Pi. The Fedora Engineering and Steering Committee (FESCo) has now signed off on this "official" support for the Raspberry Pi 4.

The Raspberry Pi 4 to date hasn't been a significant focus for Fedora Workstation due to various patches not being upstreamed-- most notably, waiting on the open-source 3D graphics bits to be upstreamed in the kernel. Now though that those upstream bits are coming together, Fedora 37 will be focusing on advertising its support for the Raspberry Pi 4 Model B as well as the Raspberry Pi 400 and Raspberry Pi CM4 compute module.

With the open-source OpenGL and Vulkan support, these latest Raspberry Pi boards are suitable for Fedora Workstation usage. The latest milestones there are V3DV just having crossed Vulkan 1.2 and Raspberry Pi 4 V3D DRM/KMS driver support in Linux 6.0. However, there still are upstream issues surrounding WiFi support for the Raspberry Pi 400, the Raspberry Pi CM4 not necessarily being very suitable for desktop use but more edge/IoT/embedded, and some device support such as around audio may be problematic.

> ***The Raspberry Pi 4 is a widely available, reasonably priced device. It has worked well in Fedora for some time in IoT and Server use cases, and now with a fully accelerated graphics stack available it's a great device from a price-per-performance perspective, and it has a wide ecosystem, so fully supporting this in Fedora makes a compelling case.***

II. .Net on JVM

1) C# on GraalVM

1.1 Wasm-based Approach

1.1.1 .Net WASI SDK

1.1.1.1 Build from source

- https://github.com/SteveSandersonMS/dotnet-wasi-sdk

First, build the runtime. This can take quite a long time.

- `git submodule update --init --recursive`
- Do the following steps using Linux or WSL:
  - `sudo apt-get install build-essential cmake ninja-build python python3 zlib1g-dev`
- `cd modules/runtime/src/mono/wasm`
  - `make provision-wasm` (takes about 2 minutes)
  - `make build-all` (takes 10-15 minutes)
    - If you get an error about `setlocale: LC_ALL: cannot change locale` then run `sudo apt install language-pack-en`. This only happens on very bare-bones machines.
- `cd ../wasi`
  - `make` (takes a few minutes - there are lots of warnings like "System is unknown to cmake" and that's OK)

Now you can build the packages and samples in this repo:

- Prerequisites
  - .NET 7 (`dotnet --version` should return `7.0.100-preview.4` or later)
  - Rust and the `wasm32-unknown-unknown` target (technically this is only needed for the CustomHost package)
    - Install Rust
    - `rustup target add wasm32-unknown-unknown`
- Just use `dotnet build` or `dotnet run` on any of the samples or `src` projects, or open the solution in VS and Ctrl+F5 on any of the sample projects

Try it on RPi4

- Build .Net WASI SDK on RPi4

```
============== start building Dotnet WASI SDK on ARM64 ==============
Cloning into 'emsdk' ...
fatal: unable to access 'https://github.com/emscripten-core/emsdk.git/': Failed to connect to github.com port 443 after 129982 ms: Connection timed out
make: *** [Makefile:37: .stamp-wasm-install-and-select-3.1.12] Error 128
2022-11-29 10:46:28 rm -rf emsdk
2022-11-29 10:46:28 git clone https://github.com/emscripten-core/emsdk.git emsdk
2022-11-29 10:48:38 EMSDK_PATH=/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/src/mono/wasm/emsdk /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/
Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/build.sh mono+libs -os Browser -c Release --binaryLog /p:ContinueOnError=false /p:StopOnFirstFailure=true
2022-11-29 10:48:38 __DistroRid: browser-wasm
2022-11-29 10:48:39 Downloading 'https://dotnet.microsoft.com/download/dotnet/scripts/v1/dotnet-install.sh'
2022-11-29 10:48:41 Attempting to install dotnet from public location.
2022-11-29 10:48:41 dotnet-install: Note that the intended use of this script is for Continuous Integration (CI) scenarios, where:
2022-11-29 10:48:41 dotnet-install: - The SDK needs to be installed without user interaction and without admin rights.
2022-11-29 10:48:41 dotnet-install: - The SDK installation doesn't need to persist across multiple CI runs.
2022-11-29 10:48:41 dotnet-install: To set up a development environment or to run apps, use installers rather than this script. Visit https://dotnet.microsoft.com/download to get the installer.
2022-11-29 10:48:41
2022-11-29 10:48:46 dotnet-install: Attempting to download using primary link https://dotnetcli.azureedge.net/dotnet/Sdk/7.0.100-preview.5.22307.18/dotnet-sdk-7.0.100-preview.5.22307.18-linux-arm64.t
ar.gz
2022-11-29 10:50:47 curl: (56) OpenSSL SSL_read: error:0A000126:SSL routines::unexpected eof while reading, errno 0
2022-11-29 10:50:49 dotnet-install: Download attempt #1 has failed: 200 Unable to download https://dotnetcli.azureedge.net/dotnet/Sdk/7.0.100-preview.5.22307.18/dotnet-sdk-7.0.100-preview.5.22307.18-
linux-arm64.tar.gz.
2022-11-29 10:50:49 dotnet-install: Attempt #2 will start in 10 seconds.
2022-11-29 10:52:30 dotnet-install: Extracting zip from https://dotnetcli.azureedge.net/dotnet/Sdk/7.0.100-preview.5.22307.18/dotnet-sdk-7.0.100-preview.5.22307.18-linux-arm64.tar.gz
2022-11-29 10:52:44 dotnet-install: Installed version is 7.0.100-preview.5.22307.18
2022-11-29 10:52:44 dotnet-install: Adding to current process PATH: `/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/.dotnet`. Note: This chan
ge will be visible only when sourcing script.
2022-11-29 10:52:44 dotnet-install: Note that the script does not resolve dependencies during installation.
2022-11-29 10:52:44 dotnet-install: To check the list of dependencies, go to https://learn.microsoft.com/dotnet/core/install, select your operating system and check the "Dependencies" section.
2022-11-29 10:52:44 dotnet-install: Installation finished successfully.
2022-11-29 10:52:45
2022-11-29 10:52:45 Welcome to .NET 7.0!
2022-11-29 10:52:45 ---------------------
2022-11-29 10:52:45 SDK Version: 7.0.100-preview.5.22307.18
```

...

```
2022-11-29 10:52:46 /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/.dotnet/sdk/7.0.100-preview.5.22307.18/MSBuild.dll /nologo -distributedlog
ger:Microsoft.DotNet.Tools.MSBuild.MSBuildLogger,/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/.dotnet/sdk/7.0.100-preview.5.22307.18/dotnet
.dll*Microsoft.DotNet.Tools.MSBuild.MSBuildForwardingLogger,/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/.dotnet/sdk/7.0.100-preview.5.2230
7.18/dotnet.dll -maxcpucount /m -verbosity:m /v:minimal /bl:/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/artifacts/log/Release/ToolsetResto
re.binlog /clp:Summary /clp:ErrorsOnly;NoSummary /nr:true /p:TreatWarningsAsErrors=true /p:ContinuousIntegrationBuild=false /p:__ToolsetLocationOutputFile=/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/D
otnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/artifacts/toolset/7.0.0-beta.22408.3.txt /t:__WriteToolsetLocation /warnaserror /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK
/Official/dotnet-wasi-sdk-main/modules/runtime/artifacts/toolset/restore.proj
2022-11-29 10:52:57 /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/.dotnet/sdk/7.0.100-preview.5.22307.18/MSBuild.dll /nologo -distributedlog
ger:Microsoft.DotNet.Tools.MSBuild.MSBuildLogger,/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/.dotnet/sdk/7.0.100-preview.5.22307.18/dotnet
.dll*Microsoft.DotNet.Tools.MSBuild.MSBuildForwardingLogger,/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/.dotnet/sdk/7.0.100-preview.5.2230
7.18/dotnet.dll -maxcpucount /m -verbosity:m /v:minimal /bl:/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/artifacts/log/Release/Build.binlog
 /clp:Summary /nr:true /p:TreatWarningsAsErrors=true /p:ContinuousIntegrationBuild=false /p:Configuration=Release /p:RepoRoot=/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotne
t-wasi-sdk-main/modules/runtime/ /p:Restore=true /p:Build=true /p:ArcadeBuildFromSource=false /p:Rebuild=false /p:Test=false /p:Pack=false /p:IntegrationTest=false /p:PerformanceTest=false /p:Sign=fa
lse /p:Publish=false /p:Subset=mono+libs /p:TargetOS=Browser /p:TargetArchitecture=wasm /p:BuildArchitecture=arm64 /p:CMakeArgs= /p:ContinueOnError=false /p:StopOnFirstFailure=true /warnaserror /home
/mydev/.nuget/packages/microsoft.dotnet.arcade.sdk/7.0.0-beta.22408.3/tools/Build.proj
2022-11-29 10:52:59   Determining projects to restore...
2022-11-29 10:53:52   Tool 'coverlet.console' (version '3.1.2') was restored. Available commands: coverlet
2022-11-29 10:53:52   Tool 'dotnet-reportgenerator-globaltool' (version '5.0.2') was restored. Available commands: reportgenerator
2022-11-29 10:53:52   Tool 'microsoft.dotnet.xharness.cli' (version '1.0.0-prerelease.22408.1') was restored. Available commands: xharness
2022-11-29 10:53:52   Tool 'microsoft.visualstudio.slngen.tool' (version '8.1.6') was restored. Available commands: slngen
2022-11-29 10:53:52   Tool 'dotnet-format' (version '6.0.240501') was restored. Available commands: dotnet-format
2022-11-29 10:53:52
2022-11-29 10:53:52   Restore was successful.
2022-11-29 10:54:33   Restored /home/mydev/.nuget/packages/microsoft.dotnet.arcade.sdk/7.0.0-beta.22408.3/tools/Tools.proj (in 40.81 sec).
2022-11-29 10:54:42   Determining projects to restore...
2022-11-29 10:56:23   Restored /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/src/libraries/System.Net.HttpListener/src/System.Net.HttpListen
er.csproj (in 17.33 sec).
2022-11-29 10:56:23   Restored /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/src/libraries/System.Net.Mail/ref/System.Net.Mail.csproj (in 17
.33 sec).
```

...

```
2022-11-29 11:04:09   WasmAppHost → /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/artifacts/bin/WasmAppHost/wasm/Release/WasmAppHost.dll
2022-11-29 11:04:17   Microsoft.NET.Runtime.WebAssembly.Sdk → /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/artifacts/packages/Release/Shipping/Microsoft.NET.Runtime.WebAssembly.Sdk.7.0.0-dev.nupkg
2022-11-29 11:04:18   Running 'bash -c 'source /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/src/mono/wasm/emsdk//emsdk_env.sh 2>&1 && emcmake cmake -G Ninja -DCMAKE_INSTALL_PREFIX="/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/artifacts/obj/mono/Browser.wasm.Release/out" -DCMAKE_INSTALL_LIBDIR=lib -DCMAKE_BUILD_TYPE=Release -DGC_SUSPEND=preemptive -DMONO_LIB_NAME=monosgen-2.0 -DMONO_SHARED_LIB_NAME=monosgen-2.0 -DENABLE_MINIMAL=jit,sgen_major_marksweep_conc,sgen_split_nursery,sgen_gc_bridge,sgen_toggleref,sgen_debug_helpers,sgen_binary_protocol,logging,interpreter,qcalls,debugger_agent,log_dest,assert_messages,threads -DENABLE_INTERP_LIB=1 -DDISABLE_ICALL_TABLES=1 -DENABLE_ICALL_EXPORT=1 -DENABLE_LAZY_GC_THREAD_CREATION=1 -DENABLE_LLVM_RUNTIME=1 -DFEATURE_PERFTRACING_DISABLE_PERFTRACING_LISTEN_PORTS=1 -DFEATURE_PERFTRACING_DISABLE_DEFAULT_LISTEN_PORT=1 -DFEATURE_PERFTRACING_DISABLE_CONNECT_PORTS=1 -DSTATIC_COMPONENTS=1 -DMONO_COMPONENTS_RID=Browser-wasm -DCMAKE_C_FLAGS=" -fexceptions -I/home/mydev/.nuget/packages/microsoft.netcore.runtime.icu.transport/7.0.0-rc.1.22408.1/runtimes/browser-wasm/native/include" -DCMAKE_CXX_FLAGS=" -fexceptions"  "/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/src/mono"'' in '/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/domake: *** [Makefile:64: build-all] Error 1
tnet-wasi-sdk-main/modules/runtime/artifacts/obj/mono/Browser.wasm.Release/'
2022-11-29 11:04:18   bash: line 1: /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/src/mono/wasm/emsdk//emsdk_env.sh: No such file or directory
2022-11-29 11:04:18 /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/src/mono/mono.proj(540,5): error MSB3073: The command "bash -c 'source /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/src/mono/wasm/emsdk//emsdk_env.sh 2>&1 && emcmake cmake -G Ninja -DCMAKE_INSTALL_PREFIX="/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/artifacts/obj/mono/Browser.wasm.Release/out" -DCMAKE_INSTALL_LIBDIR=lib -DCMAKE_BUILD_TYPE=Release -DGC_SUSPEND=preemptive -DMONO_LIB_NAME=monosgen-2.0 -DMONO_SHARED_LIB_NAME=monosgen-2.0 -DENABLE_MINIMAL=jit,sgen_major_marksweep_conc,sgen_split_nursery,sgen_gc_bridge,sgen_toggleref,sgen_debug_helpers,sgen_binary_protocol,logging,interpreter,qcalls,debugger_agent,log_dest,assert_messages,threads -DENABLE_INTERP_LIB=1 -DDISABLE_ICALL_TABLES=1 -DENABLE_ICALL_EXPORT=1 -DENABLE_LAZY_GC_THREAD_CREATION=1 -DENABLE_LLVM_RUNTIME=1 -DFEATURE_PERFTRACING_DISABLE_PERFTRACING_LISTEN_PORTS=1 -DFEATURE_PERFTRACING_DISABLE_DEFAULT_LISTEN_PORT=1 -DFEATURE_PERFTRACING_DISABLE_CONNECT_PORTS=1 -DSTATIC_COMPONENTS=1 -DMONO_COMPONENTS_RID=Browser-wasm -DCMAKE_C_FLAGS=" -fexceptions -I/home/mydev/.nuget/packages/microsoft.netcore.runtime.icu.transport/7.0.0-rc.1.22408.1/runtimes/browser-wasm/native/include" -DCMAKE_CXX_FLAGS=" -fexceptions"  "/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/src/mono"'" exited with code 1.
2022-11-29 11:04:18
2022-11-29 11:04:18 Build FAILED.
2022-11-29 11:04:18 /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/src/mono/mono.proj(540,5): error MSB3073: The command "bash -c 'source /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/src/mono/wasm/emsdk//emsdk_env.sh 2>&1 && emcmake cmake -G Ninja -DCMAKE_INSTALL_PREFIX="/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/artifacts/obj/mono/Browser.wasm.Release/out" -DCMAKE_INSTALL_LIBDIR=lib -DCMAKE_BUILD_TYPE=Release -DGC_SUSPEND=preemptive -DMONO_LIB_NAME=monosgen-2.0 -DMONO_SHARED_LIB_NAME=monosgen-2.0 -DENABLE_MINIMAL=jit,sgen_major_marksweep_conc,sgen_split_nursery,sgen_gc_bridge,sgen_toggleref,sgen_debug_helpers,sgen_binary_protocol,logging,interpreter,qcalls,debugger_agent,log_dest,assert_messages,threads -DENABLE_INTERP_LIB=1 -DDISABLE_ICALL_TABLES=1 -DENABLE_ICALL_EXPORT=1 -DENABLE_LAZY_GC_THREAD_CREATION=1 -DENABLE_LLVM_RUNTIME=1 -DFEATURE_PERFTRACING_DISABLE_PERFTRACING_LISTEN_PORTS=1 -DFEATURE_PERFTRACING_DISABLE_DEFAULT_LISTEN_PORT=1 -DFEATURE_PERFTRACING_DISABLE_CONNECT_PORTS=1 -DSTATIC_COMPONENTS=1 -DMONO_COMPONENTS_RID=Browser-wasm -DCMAKE_C_FLAGS=" -fexceptions -I/home/mydev/.nuget/packages/microsoft.netcore.runtime.icu.transport/7.0.0-rc.1.22408.1/runtimes/browser-wasm/native/include" -DCMAKE_CXX_FLAGS=" -fexceptions"  "/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/src/mono"'" exited with code 1.
2022-11-29 11:04:18     0 Warning(s)
2022-11-29 11:04:18     1 Error(s)
2022-11-29 11:04:18
2022-11-29 11:04:18 Time Elapsed 00:11:20.35
2022-11-29 11:04:19 Build failed with exit code 1. Check errors above.
2022-11-29 11:04:19 mkdir -p /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/artifacts/obj/mono/Wasi.Release
2022-11-29 11:04:19 cd /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/artifacts/obj/mono/Wasi.Release && \
2022-11-29 11:04:19     wget -q https://github.com/WebAssembly/wasi-sdk/releases/download/wasi-sdk-12/wasi-sdk-12.0-linux.tar.gz && \
2022-11-29 11:04:19     tar xf wasi-sdk-*.tar.gz
2022-11-29 11:14:11 mkdir -p /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/artifacts/obj/mono/Wasi.Release
...
System is unknown to cmake, create:
Platform/WASI to use this system, please post your config file on discourse.cmake.org so it can be added to cmake
CMake Error at /usr/share/cmake/Modules/CMakeTestCCompiler.cmake:69 (message):
  The C compiler

    "/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/artifacts/obj/mono/Wasi.Release/wasi-sdk-12.0/bin/clang"

  is not able to compile a simple test program.

  It fails with the following output:

    Change Dir: /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/artifacts/obj/mono/Wasi.Release/CMakeFiles/CMakeTmp

    Run Build Command(s):/usr/bin/ninja-build cmTC_ae85e && [1/2] Building C object CMakeFiles/cmTC_ae85e.dir/testCCompiler.c.obj
    FAILED: CMakeFiles/cmTC_ae85e.dir/testCCompiler.c.obj
    /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/artifacts/obj/mono/Wasi.Release/wasi-sdk-12.0/bin/clang   --sysroot=/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/artifacts/obj/mono/Wasi.Release/wasi-sdk-12.0/share/wasi-sysroot     -I/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/src/mono/wasi/include -I/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/src/mono -I/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/src/native/public -I/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/src/mono/mono/eglib -I/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/artifacts/obj/mono/Wasi.Release/mono/eglib -I/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/artifacts/obj/mono/Wasi.Release -I/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/artifacts/obj/wasm -I/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/src/mono/wasm/runtime -o CMakeFiles/cmTC_ae85e.dir/testCCompiler.c.obj -c /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/modules/runtime/artifacts/obj/mono/Wasi.Release/CMakeFiles/CMakeTmp/testCCompiler.c
    qemu-x86_64-static: Could not open '/lib64/ld-linux-x86-64.so.2': No such file or directory
    ninja: build stopped: subcommand failed.
...
```

We'll build the Emscripten SDK and WASI SDK on RPi4 and provide corresponding workarounds to .Net WASI SDK…

1.1.1.2 Try the prebuilt package

- https://www.nuget.org/packages/Wasi.Sdk/0.1.2-preview.10061
- e.g., cd $SRC_DotnetWasiSDK/samples/ConsoleApp

```
[mydev@fedora ConsoleApp]$ ll
total 8
drwxr-xr-x. 1 mydev mydev  68 Nov 28 11:39 ./
drwxr-xr-x. 1 mydev mydev 196 Nov 28 11:39 ../
-rw-r--r--. 1 mydev mydev 868 Nov 28 11:39 ConsoleApp.csproj
-rw-r--r--. 1 mydev mydev 155 Nov 28 11:39 Program.cs
drwxr-xr-x. 1 mydev mydev  22 Nov 28 11:39 .vscode/
[mydev@fedora ConsoleApp]$
[mydev@fedora ConsoleApp]$ bat ./Program.cs
```

```
File: ./Program.cs
1   using System.Runtime.InteropServices;
2
3   Console.WriteLine($"Hello, world at {DateTime.Now.ToLongTimeString()} on {RuntimeInformation.OSArchitecture}!");
```

```
[mydev@fedora ConsoleApp]$
[mydev@fedora ConsoleApp]$ bat ./ConsoleApp.csproj
```

```
File: ./ConsoleApp.csproj
1   <Project Sdk="Microsoft.NET.Sdk">
2
3       <!-- Only needed when referencing the dependencies as projects. For package references, these are imported automatically. -->
4       <Import Project="..\..\src\Wasi.Sdk\build\Wasi.Sdk.props" />
5
6       <PropertyGroup>
7           <OutputType>Exe</OutputType>
8           <TargetFramework>net7.0</TargetFramework>
9           <ImplicitUsings>enable</ImplicitUsings>
10          <Nullable>enable</Nullable>
11
12          <!--
13          <WasiDebugPort>64000</WasiDebugPort>
14          <WaitForDebugger>true</WaitForDebugger>
15          <DebugType>embedded</DebugType>
16          -->
17      </PropertyGroup>
18
19      <ItemGroup>
20          <ProjectReference Include="..\..\src\Wasi.Sdk\Wasi.Sdk.csproj" ReferenceOutputAssembly="false" />
21      </ItemGroup>
22
23      <!-- Only needed when referencing the dependencies as projects. For package references, these are imported automatically. -->
24      <Import Project="..\..\src\Wasi.Sdk\build\Wasi.Sdk.targets" />
25
26  </Project>
```

```
[mydev@fedora ConsoleApp]$
```

install the package of .Net WASI SDK:

```
[mydev@fedora ConsoleApp]$ dotnet add package Wasi.Sdk --prerelease
  Determining projects to restore...
  Writing /tmp/tmppd8gmX.tmp
info : X.509 certificate chain validation will use the system certificate bundle at '/etc/pki/ca-trust/extracted/pem/objsign-ca-bundle.pem'.
info : X.509 certificate chain validation will use the fallback certificate bundle at '/opt/MyWorkSpace/DevSW/DotNet/SDK/Std/7.x/sdk/7.0.100/trustedroots/timestampctl.pem'.
info : Adding PackageReference for package 'Wasi.Sdk' into project '/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/samples/ConsoleApp/ConsoleApp.csproj'.
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/wasi.sdk/index.json
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/wasi.sdk/index.json 199ms
info : Restoring packages for /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/samples/ConsoleApp/ConsoleApp.csproj ...
info :   GET https://api.nuget.org/v3-flatcontainer/wasi.sdk/index.json
info :   OK https://api.nuget.org/v3-flatcontainer/wasi.sdk/index.json 242ms
info :   GET https://api.nuget.org/v3-flatcontainer/wasi.sdk/0.1.2-preview.10061/wasi.sdk.0.1.2-preview.10061.nupkg
info :   OK https://api.nuget.org/v3-flatcontainer/wasi.sdk/0.1.2-preview.10061/wasi.sdk.0.1.2-preview.10061.nupkg 217ms
info :   GET https://api.nuget.org/v3-flatcontainer/microsoft.build.framework/index.json
info :   GET https://api.nuget.org/v3-flatcontainer/microsoft.build.utilities.core/index.json
info :   OK https://api.nuget.org/v3-flatcontainer/microsoft.build.framework/index.json 235ms
info :   GET https://api.nuget.org/v3-flatcontainer/microsoft.build.framework/17.0.0/microsoft.build.framework.17.0.0.nupkg
info :   OK https://api.nuget.org/v3-flatcontainer/microsoft.build.utilities.core/index.json 365ms
info :   OK https://api.nuget.org/v3-flatcontainer/microsoft.build.framework/17.0.0/microsoft.build.framework.17.0.0.nupkg 131ms
info :   GET https://api.nuget.org/v3-flatcontainer/microsoft.build.utilities.core/17.0.0/microsoft.build.utilities.core.17.0.0.nupkg
info :   OK https://api.nuget.org/v3-flatcontainer/microsoft.build.utilities.core/17.0.0/microsoft.build.utilities.core.17.0.0.nupkg 254ms
info : Installed Microsoft.Build.Utilities.Core 17.0.0 from https://api.nuget.org/v3/index.json with content hash +eqDvecetKfsZR9WqLQ96F9xhxFb3m9VOjkyzuaA/2D1cub1aW9XyegZb8+gEpBa+o7dHnIN9FskC+tRXtqLSQ==.
info : Installed Microsoft.Build.Framework 17.0.0 from https://api.nuget.org/v3/index.json with content hash XbFA0z+6Ws2pNeRXYcDF3lKlNgRoSGMm2Q5HKzZD+EbwYMKPKrl/BJnnkMuDJHU0KravYHfhzBnLLJpPeZ3E7A==.
info : Installed Wasi.Sdk 0.1.2-preview.10061 from https://api.nuget.org/v3/index.json with content hash OpWHXOZ0zoFJpJfQRLvpaB9pkZpwLsH5LuXQuNypyZNy9WHEZiIE1RnEbMsw+hvZyVj+tMERSkEyfHtMfRU7qQ==.
info : Package 'Wasi.Sdk' is compatible with all the specified frameworks in project '/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/samples/ConsoleApp/ConsoleApp.csproj'.
info : PackageReference for package 'Wasi.Sdk' version '0.1.2-preview.10061' added to file '/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/samples/ConsoleApp/ConsoleApp.csproj'.
info : Generating MSBuild file /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/samples/ConsoleApp/obj/ConsoleApp.csproj.nuget.g.props.
info : Generating MSBuild file /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/samples/ConsoleApp/obj/ConsoleApp.csproj.nuget.g.targets.
info : Writing assets file to disk. Path: /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/samples/ConsoleApp/obj/project.assets.json
log  : Restored /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/samples/ConsoleApp/ConsoleApp.csproj (in 6.38 sec).
[mydev@fedora ConsoleApp]$
```

```
[mydev@fedora ConsoleApp]$ tree -L 5 /home/mydev/.nuget/packages/wasi.sdk
/home/mydev/.nuget/packages/wasi.sdk
└── 0.1.2-preview.10061
    ├── build
    │   ├── Wasi.Sdk.props
    │   └── Wasi.Sdk.targets
    ├── native
    │   └── main.c
    ├── packs
    │   └── wasi-wasm
    │       ├── lib
    │       │   └── net7.0
    │       └── native
    │           ├── include
    │           ├── libmono-component-debugger-static.a
    │           ├── libmono-component-debugger-stub-static.a
    │           ├── libmono-component-diagnostics_tracing-stub-static.a
    │           ├── libmono-component-hot_reload-stub-static.a
    │           ├── libmono-component-marshal-ilgen-static.a
    │           ├── libmono-component-marshal-ilgen-stub-static.a
    │           ├── libmono-ee-interp.a
    │           ├── libmono-ilgen.a
    │           ├── libmonosgen-2.0.a
    │           ├── libmono-wasi-driver.a
    │           └── libSystem.Native.a
    ├── tools
    │   ├── net7.0
    │   │   └── Wasi.Sdk.dll
    │   └── netstandard2.0
    │       ├── System.Buffers.dll
    │       ├── System.Collections.Immutable.dll
    │       ├── System.Memory.dll
    │       ├── System.Numerics.Vectors.dll
    │       ├── System.Reflection.Metadata.dll
    │       ├── System.Runtime.CompilerServices.Unsafe.dll
    │       └── Wasi.Sdk.dll
    ├── wasi.sdk.0.1.2-preview.10061.nupkg
    ├── wasi.sdk.0.1.2-preview.10061.nupkg.sha512
    └── wasi.sdk.nuspec
```

initial build:

```
[mydev@fedora samples]$ git diff
diff --git a/samples/ConsoleApp/ConsoleApp.csproj b/samples/ConsoleApp/ConsoleApp.csproj
index 36201eb..0334eff 100644
--- a/samples/ConsoleApp/ConsoleApp.csproj
+++ b/samples/ConsoleApp/ConsoleApp.csproj
@@ -20,6 +20,10 @@
                <ProjectReference Include="..\..\src\Wasi.Sdk\Wasi.Sdk.csproj" ReferenceOutputAssembly="false" />
        </ItemGroup>

+       <ItemGroup>
+         <PackageReference Include="Wasi.Sdk" Version="0.1.2-preview.10061" />
+       </ItemGroup>
+
        <!-- Only needed when referencing the dependencies as projects. For package references, these are imported automatically. -->
        <Import Project="..\..\src\Wasi.Sdk\build\Wasi.Sdk.targets" />

[mydev@fedora samples]$
```

```
[mydev@fedora ConsoleApp]$ dotnet build
MSBuild version 17.4.0+18d5aef85 for .NET
  Determining projects to restore...
  Restored /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/src/Wasi.Sdk/Wasi.Sdk.csproj (in 1.49 sec).
  1 of 2 projects are up-to-date for restore.
  Wasi.Sdk → /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/src/Wasi.Sdk/tools/net7.0/Wasi.Sdk.dll
/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/src/Wasi.Sdk/Wasi.Sdk.csproj(55,3): error MSB3030: Could not copy the file "../../modules/runtime/artifacts/bi
n/microsoft.netcore.app.runtime.browser-wasm/Release/runtimes/browser-wasm/native/System.Private.CoreLib.dll" because it was not found. [TargetFramework=net7.0]

Build FAILED.

/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/src/Wasi.Sdk/Wasi.Sdk.csproj(55,3): error MSB3030: Could not copy the file "../../modules/runtime/artifacts/bi
n/microsoft.netcore.app.runtime.browser-wasm/Release/runtimes/browser-wasm/native/System.Private.CoreLib.dll" because it was not found. [TargetFramework=net7.0]
    0 Warning(s)
    1 Error(s)

Time Elapsed 00:00:23.03
[mydev@fedora ConsoleApp]$
```

vim ../../src/Wasi.Sdk/Wasi.Sdk.csproj

```
<Project Sdk="Microsoft.NET.Sdk">

        <PropertyGroup>
                <TargetFrameworks>net7.0;netstandard2.0</TargetFrameworks>
                <LangVersion>latest</LangVersion>
                <GenerateDependencyFile>false</GenerateDependencyFile>
                <CopyLocalLockFileAssemblies Condition="'$(TargetFramework)' == 'netstandard2.0'">true</CopyLocalLockFileAssemblies>

                <!-- Suppresses the warnings about the package not having assemblies in lib/*/.dll. -->
        <NoPackageAnalysis>true</NoPackageAnalysis>

                <!-- OutDir is used in regular builds; BuildOutputTargetFolder is used when packing -->
                <OutDir>tools\$(TargetFramework)\</OutDir>
                <BuildOutputTargetFolder>tools</BuildOutputTargetFolder>
                <Nullable>enable</Nullable>
        </PropertyGroup>

        <ItemGroup>
                <PackageReference Include="Microsoft.Build.Framework" Version="17.0.0" ExcludeAssets="Runtime" />
                <PackageReference Include="Microsoft.Build.Utilities.Core" Version="17.0.0" ExcludeAssets="Runtime" />
                <PackageReference Include="System.Reflection.Metadata" Version="6.0.0" Condition="'$(TargetFramework)' == 'netstandard2.0'" />
        </ItemGroup>

        <Target Name="PrepareWasiWasmPack" BeforeTargets="Build">
                <!--
                        These files become available when building modules/runtime

                        - Use WSL
                        - cd modules/runtime/src/mono/wasm
                        - make build-all
                        - cd ../wasi
                        - make
                -->
                <PropertyGroup>
                        <_WasiRuntimeArtifactsBin>..\..\modules\runtime\artifacts\bin\</_WasiRuntimeArtifactsBin>
                        <_WasiRuntimeArtifactsNativeRoot>$(_WasiRuntimeArtifactsBin)mono\Wasi.Release\</_WasiRuntimeArtifactsNativeRoot>
                        <_WasiRuntimeArtifactsBrowserWasmRoot>$(_WasiRuntimeArtifactsBin)microsoft.netcore.app.runtime.browser-wasm\Release\runtimes\browser-wasm\</_WasiRuntimeArtifactsBrowserWasmRoot>
                </PropertyGroup>
                <ItemGroup>
                        <_WasiPackLibFiles Include="$(_WasiRuntimeArtifactsBrowserWasmRoot)lib\**\*.dll" />
                        <_WasiPackNativeFiles Include="$(_WasiRuntimeArtifactsNativeRoot)**" />

                        <!-- Grab the .h files from browser-wasm -->
                        <!-- TODO: Amend the Wasi.Release build output to include these -->
                        <_WasiPackNativeFiles Include="$(_WasiRuntimeArtifactsBrowserWasmRoot)native\include\mono-2.0\**" Subdir="include\" />
                        <_WasiPackNativeFiles Include="$(_WasiRuntimeArtifactsBrowserWasmRoot)native\include\wasm\**" Subdir="include\" />

                        <!-- Workaround for using libSystem.Native.a from the browser-wasm build output -->
                        <!-- TODO: Build this using WASI SDK, and get the output from _WasiRuntimeArtifactsNativeRoot -->
                        <_WasiPackNativeFiles Include="$(_WasiRuntimeArtifactsBrowserWasmRoot)\native\libSystem.Native.a" />

                        <!-- Not sure why we need to get this from the browser-wasm build output -->
                        <_WasiPackLibFiles Include="$(_WasiRuntimeArtifactsBrowserWasmRoot)native\System.Private.CoreLib.dll" Subdir="net7.0" />
                </ItemGroup>
                <Copy SourceFiles="@(_WasiPackLibFiles)" DestinationFolder="packs\wasi-wasm\lib\%(_WasiPackLibFiles.Subdir)%(RecursiveDir)" SkipUnchangedFiles="true" />
                <Copy SourceFiles="@(_WasiPackNativeFiles)" DestinationFolder="packs\wasi-wasm\native\%(_WasiPackNativeFiles.Subdir)%(RecursiveDir)" SkipUnchangedFiles="true" />
        </Target>

        <Target Name="PackTaskDependencies" BeforeTargets="GenerateNuspec">
                <!-- From https://natemcmaster.com/blog/2017/11/11/msbuild-task-with-dependencies/#third-party-dependencies-and-nuget -->
                <!-- For .NET Framework, we need to include dependency assemblies in the tools dir. -->
                <ItemGroup>
                        <_PackageFiles Include="$(OutDir)*\System*.dll">
                                <PackagePath>tools\%(RecursiveDir)</PackagePath>
                                <BuildAction>Content</BuildAction>
                        </_PackageFiles>
                </ItemGroup>

                <!-- Also include other required files -->
                <ItemGroup>
                        <_PackageFiles Include="build\**" BuildAction="Content" PackagePath="build" />
                        <_PackageFiles Include="native\**" BuildAction="Content" PackagePath="native" />
                        <_PackageFiles Include="packs\**" BuildAction="Content" PackagePath="packs" />
                </ItemGroup>
        </Target>

</Project>
```

specific the System.Private.CoreLib.dll and libSystem.Native.a for rebuild:

```
[mydev@fedora ConsoleApp]$ git diff
diff --git a/samples/ConsoleApp/ConsoleApp.csproj b/samples/ConsoleApp/ConsoleApp.csproj
index 36201eb..0334eff 100644
--- a/samples/ConsoleApp/ConsoleApp.csproj
+++ b/samples/ConsoleApp/ConsoleApp.csproj
@@ -20,6 +20,10 @@
                <ProjectReference Include="..\..\src\Wasi.Sdk\Wasi.Sdk.csproj" ReferenceOutputAssembly="false" />
        </ItemGroup>

+       <ItemGroup>
+         <PackageReference Include="Wasi.Sdk" Version="0.1.2-preview.10061" />
+       </ItemGroup>
+
        <!-- Only needed when referencing the dependencies as projects. For package references, these are imported automatically. -->
        <Import Project="..\..\src\Wasi.Sdk\build\Wasi.Sdk.targets" />

diff --git a/src/Wasi.Sdk/Wasi.Sdk.csproj b/src/Wasi.Sdk/Wasi.Sdk.csproj
index 287ee9b..f14b8d0 100644
--- a/src/Wasi.Sdk/Wasi.Sdk.csproj
+++ b/src/Wasi.Sdk/Wasi.Sdk.csproj
@@ -47,10 +47,10 @@

                        <!-- Workaround for using libSystem.Native.a from the browser-wasm build output -->
                        <!-- TODO: Build this using WASI SDK, and get the output from _WasiRuntimeArtifactsNativeRoot -->
-                       <_WasiPackNativeFiles Include="$(_WasiRuntimeArtifactsBrowserWasmRoot)\native\libSystem.Native.a" />
+                       <_WasiPackNativeFiles Include="/home/mydev/.nuget/packages/wasi.sdk/0.1.2-preview.10061/packs/wasi-wasm/native/libSystem.Native.a" />

                        <!-- Not sure why we need to get this from the browser-wasm build output -->
-                       <_WasiPackLibFiles Include="$(_WasiRuntimeArtifactsBrowserWasmRoot)native\System.Private.CoreLib.dll" Subdir="net7.0" />
+                       <_WasiPackLibFiles Include="..\..\modules\runtime\.dotnet\shared\Microsoft.NETCore.App\7.0.0-preview.5.22301.12\System.Private.CoreLib.dll" Subdir="net7.0" />
                </ItemGroup>
                <Copy SourceFiles="@(_WasiPackLibFiles)" DestinationFolder="packs\wasi-wasm\lib\%(_WasiPackLibFiles.Subdir)%(RecursiveDir)" SkipUnchangedFiles="true" />
                <Copy SourceFiles="@(_WasiPackNativeFiles)" DestinationFolder="packs\wasi-wasm\native\%(_WasiPackNativeFiles.Subdir)%(RecursiveDir)" SkipUnchangedFiles="true" />
[mydev@fedora ConsoleApp]$
```

```
[mydev@fedora ConsoleApp]$ dotnet build
MSBuild version 17.4.0+18d5aef85 for .NET
  Determining projects to restore...
  All projects are up-to-date for restore.
  Wasi.Sdk → /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/src/Wasi.Sdk/tools/net7.0/Wasi.Sdk.dll
  ConsoleApp → /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/samples/ConsoleApp/bin/Debug/net7.0/ConsoleApp.dll
  Downloading from "https://github.com/WebAssembly/wasi-sdk/releases/download/wasi-sdk-16/wasi-sdk-16.0-linux.tar.gz" to "/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/samples/ConsoleApp/obj/Debug/net7.0/wasi-sdk-temp/wasi-sdk-16.0-linux.tar.gz" (66,173,204 bytes).
  Extracting /opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/samples/ConsoleApp/obj/Debug/net7.0/wasi-sdk-temp/wasi-sdk-16.0-linux.tar.gz to /home/mydev/.wasi-sdk/wasi-sdk-16.0...
  2/14 Bundling System.Runtime.dll...
  1/14 Bundling System.Threading.ThreadPool.dll...
  4/14 Bundling System.Threading.Channels.dll...
  3/14 Bundling ConsoleApp.dll...
qemu-x86_64-static: Could not open '/lib64/ld-linux-x86-64.so.2': No such file or directory
qemu-x86_64-static: Could not open '/lib64/ld-linux-x86-64.so.2': No such file or directory
qemu-x86_64-static: Could not open '/lib64/ld-linux-x86-64.so.2': No such file or directory
  5/14 Bundling System.Console.dll...
  6/14 Bundling System.Collections.dll...
qemu-x86_64-static: Could not open '/lib64/ld-linux-x86-64.so.2': No such file or directory
qemu-x86_64-static: Could not open '/lib64/ld-linux-x86-64.so.2': No such file or directory
qemu-x86_64-static: Could not open '/lib64/ld-linux-x86-64.so.2': No such file or directory
/home/mydev/.nuget/packages/wasi.sdk/0.1.2-preview.10061/build/Wasi.Sdk.targets(170,3): error MSB4018: The "EmitWasmBundleObjectFile" task failed unexpectedly. [/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/samples/ConsoleApp/ConsoleApp.csproj]
/home/mydev/.nuget/packages/wasi.sdk/0.1.2-preview.10061/build/Wasi.Sdk.targets(170,3): error MSB4018: System.AggregateException: One or more errors occurred. (Pipe is broken.) (Pipe is broken.) (Pipe is broken.) (Pipe is broken.) [/opt/MyWorkSpace/MyProjs/Runtime/WASM/WASI/Dotnet-WASI-SDK/Official/dotnet-wasi-sdk-main/samples/ConsoleApp/ConsoleApp.csproj]
...
```

We'll try it after build the Emscripten SDK and WASI SDK on RPi4 and provide corresponding workarounds to .Net WASI SDK

1.1.2 Mono

1.1.2.1 mono-wasm

- https://www.mono-project.com/news/2018/01/16/mono-static-webassembly-compilation/

WebAssembly static compilation in Mono is orchestrated with the `mono-wasm` command-line tool. This program takes IL assemblies as input and generates a series of files in an output directory, notably an `index.wasm` file containing the WebAssembly code for your assemblies as well as all other dependencies (the Mono runtime, the C library and the `mscorlib.dll` library).

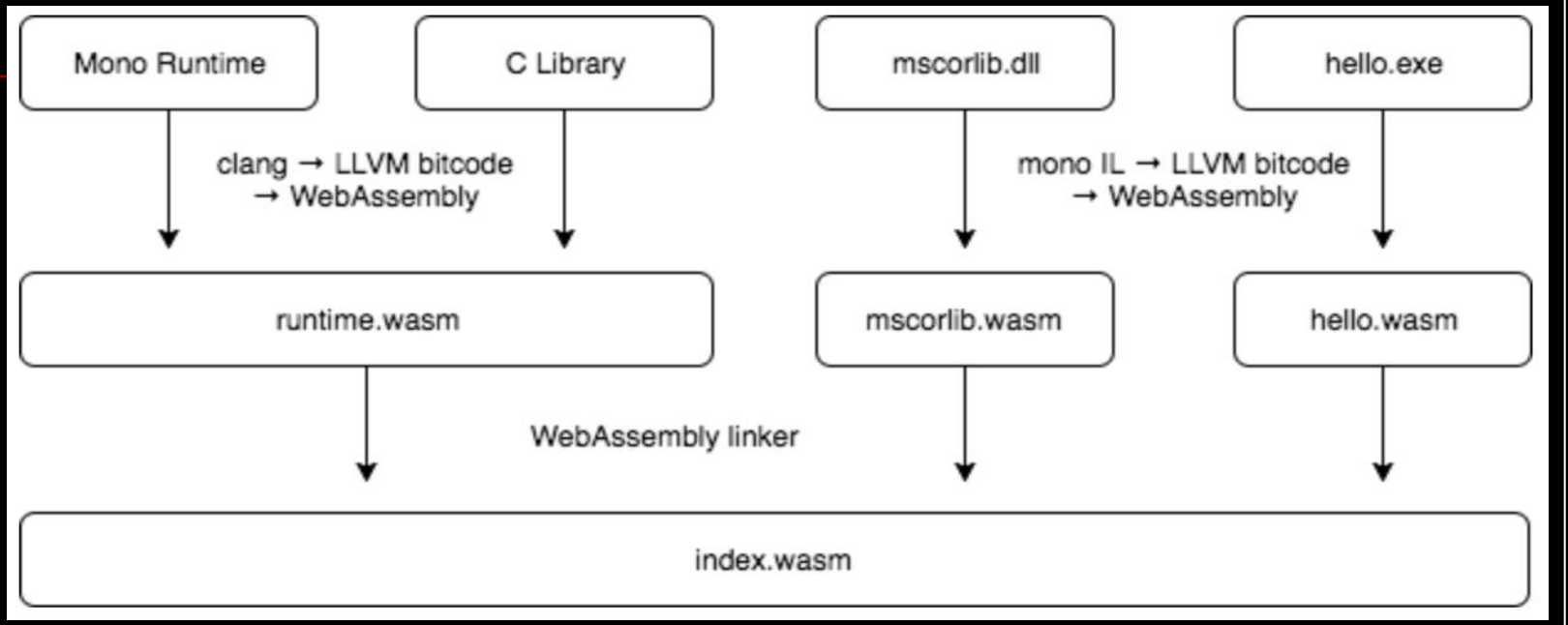
```
$ cat hello.cs
class Hello {
  static int Main(string[] args) {
    System.Console.WriteLine("hello world!");
    return 0;
  }
}
$ mcs -nostdlib -noconfig -r:../../dist/lib/mscorlib.dll hello.cs -out:hello.exe
$ mono-wasm -i hello.exe -o output
$ ls output
hello.exe        index.html        index.js        index.wasm        mscorlib.dll
```

`mono-wasm` uses a version of the Mono compiler that, given C# assemblies, generates LLVM bitcode suitable to be passed to the LLVM WebAssembly backend. Similarly, we have been building the Mono runtime and a C library with a version of `clang` that also generates LLVM WebAssembly bitcode.

`mono-wasm` uses a version of the Mono compiler that, given C# assemblies, generates LLVM bitcode suitable to be passed to the LLVM WebAssembly backend. Similarly, we have been building the Mono runtime and a C library with a version of `clang` that also generates LLVM WebAssembly bitcode.

Until recently, `mono-wasm` was linking all the bitcode into a single LLVM module then performing the WebAssembly code generation on it. While this created a functional `.wasm` file, this had the downside of taking a significant amount of time (half a minute on a recent MacBook Pro) every time we were building a project as a lot of code was in play. Some of the code, the runtime bits and the `mscorlib.dll` library, never changed and yet were still being processed for WebAssembly code generation every time.

Since then, we changed our `mono-wasm` tool to perform incremental compilation of project dependencies into separate `.wasm` files, and we integrated `lld`'s new WebAssembly driver in the tool. Thanks to this approach, we now perform WebAssembly code generation only when required, and in our testing builds now complete in less than a second once the dependencies (runtime bits and `mscorlib.dll`) have already been compiled into WebAssembly.

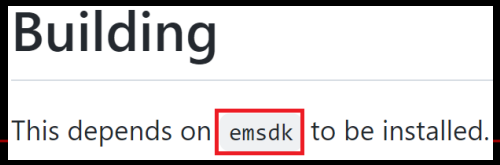


Additionally, `mono-wasm` used to use the LLVM WebAssembly target to create source files that would then be passed to the Binaryen toolchain to create the `.wasm` code. We have been testing the backend's ability to generate `.wasm` object files directly (with the `wasm32-unknown-unknown-wasm` triple) and so far it seems promising enough that we changed `mono-wasm` accordingly. We also noticed a slight decrease in build time.

- **https://github.com/migueldeicaza/mono-wasm**
- **…**

1.1.2.2 .Net Runtime

- https://github.com/dotnet/runtime/tree/main/src/mono/wasm




```
[mydev@fedora runtime]$ tree -L 1 src/mono/wasm
src/mono/wasm
├── build
├── data
├── debugger
├── emscripten-version.txt
├── host
├── Makefile
├── README.md
├── runtime
├── sanitize.py
├── sln
├── symbolicator
├── templates
├── testassets
├── test-main.js
├── threads.md
├── Wasm.Build.Tests
├── wasm.code-workspace
└── wasm.proj
```

```
[mydev@fedora runtime]$ tree src/mono/wasm/build
src/mono/wasm/build
├── EmSdkRepo.Defaults.props
├── README.md
├── WasmApp.InTree.props
├── WasmApp.InTree.targets
├── WasmApp.LocalBuild.props
├── WasmApp.LocalBuild.targets
├── WasmApp.Native.targets
├── WasmApp.props
└── WasmApp.targets
```

```
[mydev@fedora runtime]$ tree -L 1 src/mono/sample/wasm
src/mono/sample/wasm
├── browser
├── browser-advanced
├── browser-bench
├── browser-eventpipe
├── browser-nextjs
├── browser-profile
├── browser-threads
├── browser-webpack
├── CommonAssemblyInfo.cs
├── console-node
├── console-node-ts
├── console-v8
├── DefaultBrowserSample.targets
├── Directory.Build.props
├── Directory.Build.targets
├── node-webpack
├── simple-raytracer
├── simple-server
└── wasm.mk
```

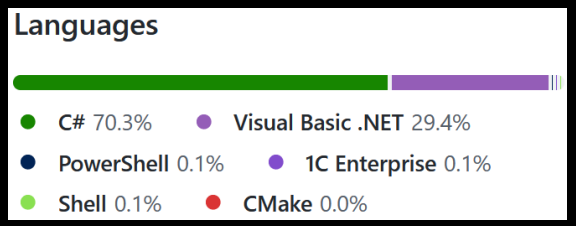
- https://github.com/dotnet/runtime/blob/main/docs/design/mono/wasm-aot.md
- https://github.com/dotnet/runtime/blob/main/docs/workflow/testing/libraries/testing-wasm.md
- ...

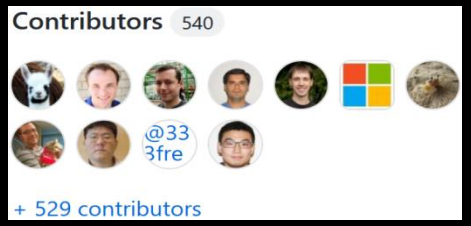
1.1.3 Roslyn

1.1.3.1 Overview

- https://en.wikipedia.org/wiki/Roslyn_(compiler)
- https://github.com/dotnet/roslyn

Roslyn is the open-source implementation of both the C# and Visual Basic compilers with an API surface for building code analysis tools.





- https://learn.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/compiler-api-model
- https://github.com/dotnet/roslyn/blob/main/docs/wiki/Roslyn-Overview.md
- https://github.com/dotnet/runtime/issues/77610

Is it possible for roslyn to support wasm as compiling target?

III. Java on .Net

…

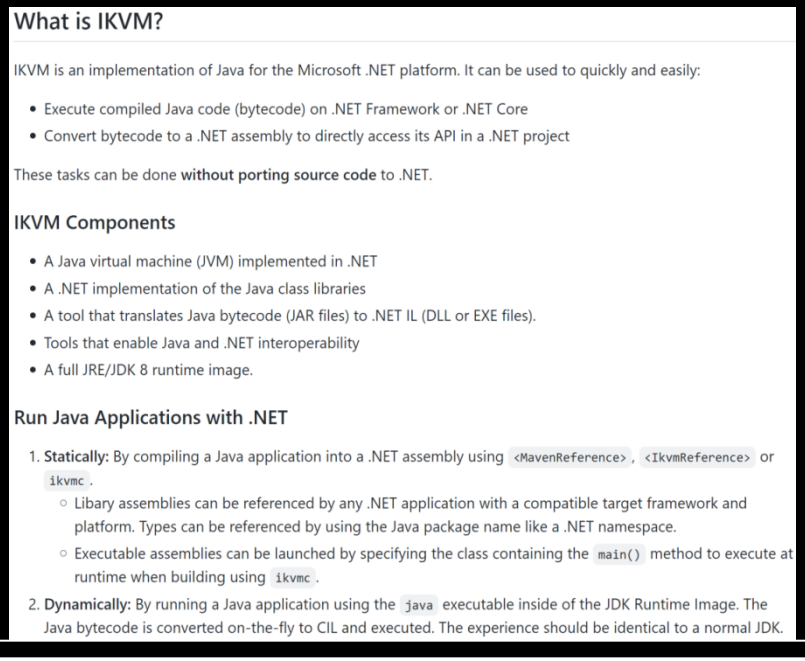
- …

1) Bytecode Conversion
1.1 IKVM
1.1.1 Overview

- https://en.wikipedia.org/wiki/IKVM.NET
- https://github.com/ikvm-revived/ikvm

A Java Virtual Machine and Bytecode-to-IL Converter for .NET


```
[submodule "openjdk"]
        path = openjdk
        url = https://github.com/ikvm-revived/jdk8u.git
        ignore = dirty
[submodule "jtreg"]
        path = jtreg
        url = https://github.com/ikvm-revived/jtreg.git
        ignore = dirty
```

- 

What is IKVM?

IKVM is an implementation of Java for the Microsoft .NET platform. It can be used to quickly and easily:

- Execute compiled Java code (bytecode) on .NET Framework or .NET Core
- Convert bytecode to a .NET assembly to directly access its API in a .NET project

These tasks can be done **without porting source code** to .NET.

IKVM Components

- A Java virtual machine (JVM) implemented in .NET
- A .NET implementation of the Java class libraries
- A tool that translates Java bytecode (JAR files) to .NET IL (DLL or EXE files).
- Tools that enable Java and .NET interoperability
- A full JRE/JDK 8 runtime image.

Run Java Applications with .NET

1. **Statically:** By compiling a Java application into a .NET assembly using `<MavenReference>`, `<IkvmReference>` or `ikvmc`.
   - Libary assemblies can be referenced by any .NET application with a compatible target framework and platform. Types can be referenced by using the Java package name like a .NET namespace.
   - Executable assemblies can be launched by specifying the class containing the `main()` method to execute at runtime when building using `ikvmc`.
2. **Dynamically:** By running a Java application using the `java` executable inside of the JDK Runtime Image. The Java bytecode is converted on-the-fly to CIL and executed. The experience should be identical to a normal JDK.

What IKVM is Not

- A converter utility to transform Java source code to C# source code
- A decompiler utitity to transform compiled Java bytecode to C# source code
- A tool that runs .NET code in Java - all IKVM conversions are Java > .NET

IV. Wasm on JVM

...

- ...

V. Java on Wasm

…

- …

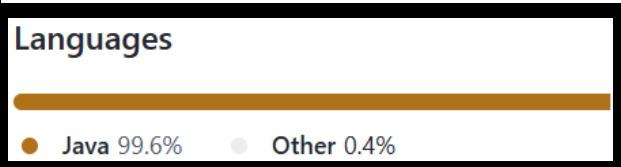
1) Bytecode Conversion
1.1 JWebAssembly
1.1.1 Overview

- https://github.com/i-net-software/JWebAssembly

Java bytecode to WebAssembly compiler.

JWebAssembly is a Java bytecode to WebAssembly compiler. It uses Java class files as input. That it can compile any language that compile to Java bytecode like Clojure, Groovy, JRuby, Jython, Kotlin and Scala. As output it generates the binary format (.wasm file) or the text format (.wat file). The target is to run Java natively in the browser with WebAssembly.

The difference to similar projects is that not a complete VM with GC and memory management should be ported. It's more like a 1: 1 conversion. The generated WebAssembly code is similar in size to the original Java class files.



- https://github.com/i-net-software/JWebAssembly/wiki
- …

Usage

Exporting functions

To export a Java function to make it accessible from JavaScript, you must add the annotation de.inetsoftware.jwebassembly.api.annotation.Export.

```
import de.inetsoftware.jwebassembly.api.annotation.Export;

@Export
public static int add( int a, int b ) {
    return a + b;
}
```

importing functions

To import a JavaScript function to make it accessible from Java, you must add the annotation de.inetsoftware.jwebassembly.api.annotation.Import. The method can be declared native or can have a Java implementation which will be ignored on compiling.

```
import de.inetsoftware.jwebassembly.api.annotation.Import;

@Import( module = "global.Math", name = "max" )
static int max( int a, int b) {
    return Math.max( a, b );
}
```

Hello world sample in the browser with the DOM wrapper API

```
@Export
public static void main() {
    Document document = Window.document();
    HTMLElement div = document.createElement("div");
    Text text = document.createTextNode("Hello World, this text come from WebAssembly.");
    div.appendChild( text );
    document.body().appendChild( div );
}
```

Source: https://github.com/i-net-software/JWebAssembly

1.1.1.1 Status

Status of Required WebAssembly Features

The following table shows the status of future WebAssembly features required by JWebAssembly in nightly builds in various implementations. These features are already used by the trunk version of JWebAssembly. If you want know the status of your current browser then look for your browser support.

Feature	Importance	V8/Chrome	SpiderMonkey/FF	WABT
Garbage collection	medium	-	partly	-
Exceptions	low	partly	-	partly
Threads	low	yes	?	yes
Tail call	very low	yes	?	yes

- For V8 it based on the V8 - node.js integrations builds.
- For SpiderMonkey it based on the nightly build of jsshell.
- For WABT it based on libwabt.js via node module wabt@nightly.

To use it also some flags and switches are currently needed.

Importance: All with high marked features are required for a hello word sample. For a first version that can be used for production.

1.1.1.2 Roadmap

- The compiler is feature complete for milestone 1. There is a release candidate. Please test it and report bugs if the compiled code has the same behavior as Java.

  Also the current browsers (Chrome, Edge, Firefox, Opera, ...) supports the needed features.

- **Version 1.0 (Milestone 1)**

Desired Features

- [x] Java byte code parser
- [x] test framework
- [x] Public API of the Compiler see class JWebAssembly
- [x] Gradle Plugin
- [x] Emulator
- [x] Binary format file writer and Text format file writer
- [x] Support for native methods #2
- [x] Memory Management - currently with a polyfill on JavaScript side
- [x] invoke static method calls
- [x] invoke instance method calls
- [x] invoke interface method calls
- [x] invoke dynamic method calls (lambdas)
- [x] invoke default method calls
- [x] String support
- [x] Simple Class object support
- [x] static constructors
- [x] Enum support
- [x] Optimizer - Optimize the WASM output of a single method after transpiling before writing to output
- [x] Hello World sample (live), (source code)
- [x] Collection framework compile

- **Version 2.0 (Milestone 2)**

Desired Features

- [ ] Full featured library for accessing the DOM.
- [ ] Embbedding of Resources (properties, images, etc.)

- **Version 3.0 (Milestone 3)**

Desired Features

- [ ] Exception handling - required the next version of WebAssembly
- [ ] Multiple threads - required the next version of WebAssembly
- [ ] Memory Management with built-in GC without JavaScript polyfill
- [ ] Reflection
- [ ] More optimize like tail calls, removing from asserts, inlining of functions, etc

2) Combination
2.1 TeaVM
2.1.1 Overview

- https://teavm.org/

TeaVM is an ahead-of-time compiler for Java bytecode that emits JavaScript and WebAssembly that runs in a browser. Its close relative is the well-known GWT. The main difference is that TeaVM does not require source code, only compiled class files. Moreover, the source code is not required to be Java, so TeaVM successfully compiles Kotlin and Scala.

- Features

Easy to use

Create a new project from a Maven archetype and develop with pleasure. You don't need a complicated setup with npm, Webpack, UglifyJS, Babel, etc. TeaVM does everything for you.

A powerful web framework included

With TeaVM, you can use Flavour, a framework for creating single-page web applications in Java, Kotlin or Scala. The experience is quite close to Angular, but the framework is based on Java idioms, not JavaScript idioms.

Efficient

TeaVM is quite efficient, you won't wait for hours while it's compiling. Also, TeaVM produces fast, small JavaScript code for web apps that start quickly, even on mobile devices.

- https://teavm.org/docs/intro/overview.html
- https://github.com/konsoletyper/teavm/wiki

Motivation

Why use TeaVM when there are plenty of transpilers and frameworks for web front-end development?

If you are a JavaScript developer who is satisfied with JavaScript, TypeScript or even elm, you probably won't need TeaVM.

If you are a Java (or Kotlin, or Scala) developer who is used to writing back-end code, TeaVM might be your choice. It's true that a good developer (including Java developer) can learn JavaScript. However, to become an expert you have to spend a reasonable amount of your time.

Another drawback of JavaScript for you is it's a different language with different syntax, different build tools, different IDE experience. Say, you have your Java/Maven build set up in Jenkins, with SonarQube checking your code quality and IDEA code style settings. You have to repeat all these things for the JavaScript ecosystem. Finally, you have to "switch context" every time you change the code on back-end and front-end side.

TeaVM allows you to use single ecosystem, and reuse as much as possible of it for both back-end and front-end worlds.

Purpose

TeaVM is primarily a web development tool. It's not for taking your large existing codebase in Java or Kotlin and producing JavaScript. Unfortunately, Java was not designed to run efficiently in the browser. There are Java APIs that are impossible to implement without generating inefficient JavaScript. Some of these APIs are: reflection, resources, class loaders, and JNI. TeaVM restricts usage of these APIs. Generally, you'll have to manually rewrite your code to fit into TeaVM constraints.

TeaVM is for you, if:

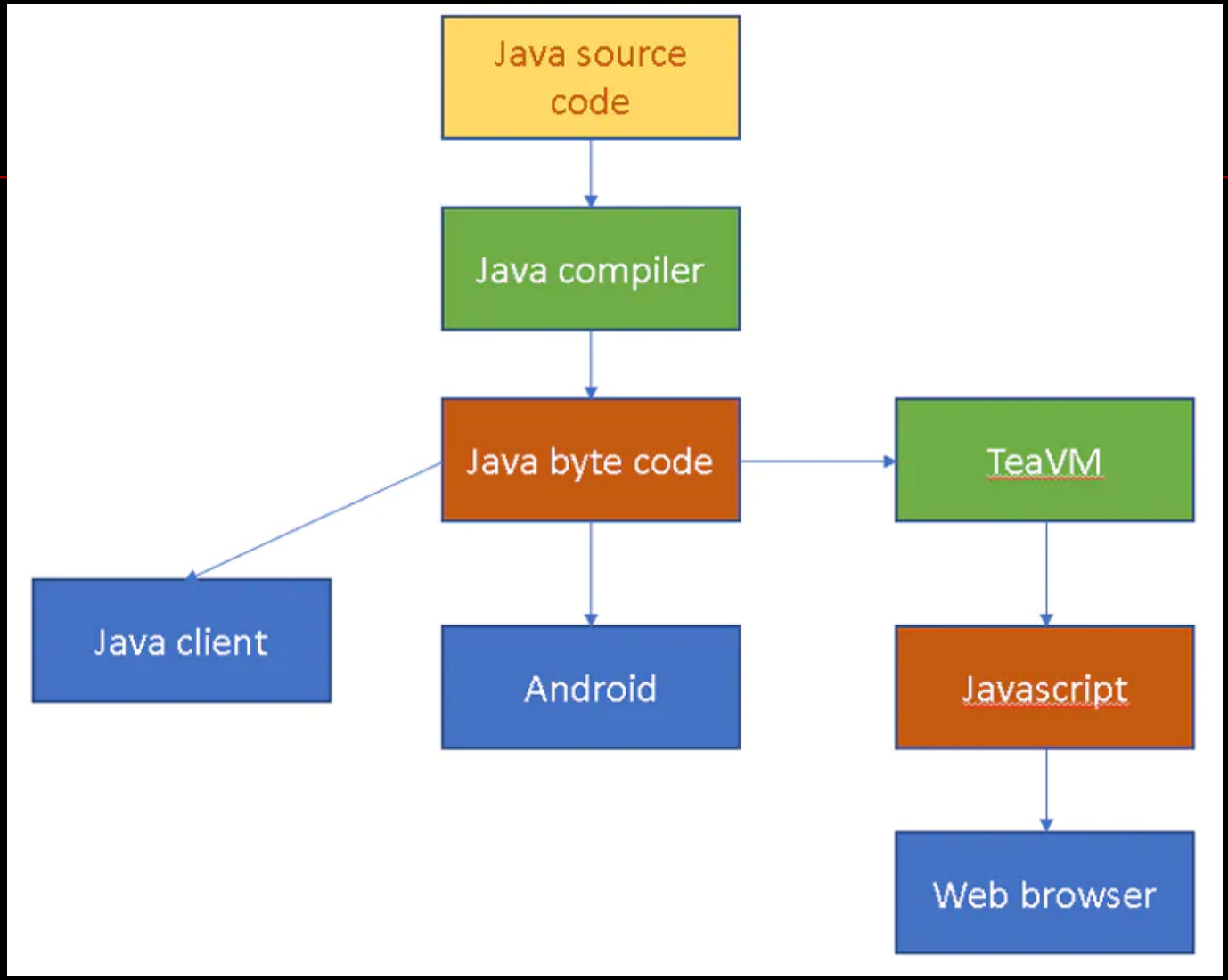
- You are a Java developer and you are going to write a web front-end from scratch.
- You already have a Java-based backend and want to integrate front-end code tightly into your existing development infrastructure.
- You have some Java back-end code you want to reuse in the front end.
- You are ready to rewrite your code to work with TeaVM.

If you have tightly-coupled applications that use Swing, you want to run these applications in web, and you don't care about download size, start-up time and performance, you should probably look elsewhere; there are more appropriate tools for you, like CheerpJ.

Strong parts

- TeaVM tries to reconstruct original structure of a method, so in most cases it produces JavaScript that you would write manually. No bloated while/switch statements, as naive compilers often do.
- TeaVM has a very sophisticated optimizer, which knows a lot about your code. Some examples are:
  - Dead code elimination produces very small JavaScript.
  - Devirtualization turns virtual calls into static function calls, which makes code faster.
  - TeaVM can reuse one local variables to store several local variables.
  - TeaVM renames methods to as short forms as possible; UglifyJS usually can't perform such optimization.
- TeaVM supports threads. JavaScript does not provide APIs to create threads (WebWorkers are not threads, since they don't allow to share state between workers). TeaVM is capable of transforming methods to continuation-passing style. This makes possible to emulate multiple logical threads in one physical thread. TeaVM threads are, in fact, green threads.
- TeaVM is very fast, you don't need to wait for minutes until your application gets recompiled.
- TeaVM produces source maps; TeaVM IDEA plugin allows you to debug code right from the IDE.
- TeaVM has a nice JavaScript interop API.

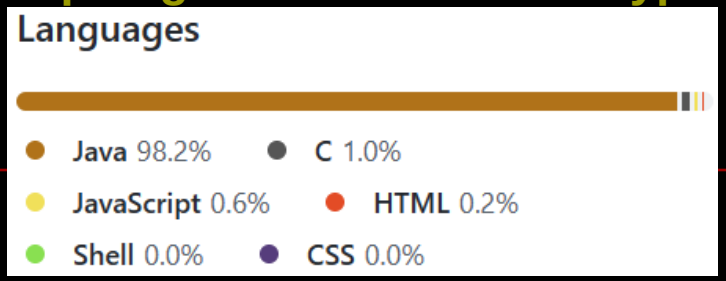
Workflow

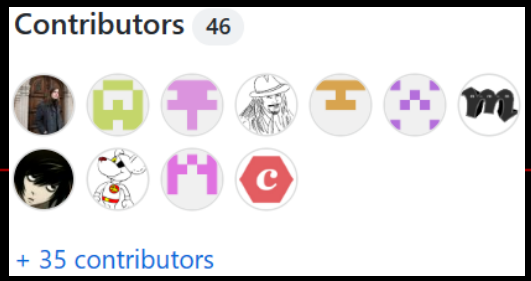


Source: https://www.electronicdesign.com/blogs/altembedded/article/21146296/electronic-design-transpilers-emulators-and-containers-oh-my

Src

- https://github.com/konsoletyper/teavm





Embedding TeaVM

If you are not satisfied with Maven, you can embed TeaVM in your program or even create your own plugin for any build tool, like Ant or Gradle. The starting point for you may be `org.teavm.tooling.TeaVMTool` class from `teavm-tooling` artifact. You may want to go deeper and use `org.teavm.vm.TeaVM` from `teavm-core` artifact, learn how `TeaVMTool` initializes it. To learn how to use `TeaVMTool` class itself, find its usages across project source code. You most likely encounter Maven and IDEA plugins.

Please, notice that these APIs for embedding are still unstable and may change between versions.

2.1.1.1 Flavour framework

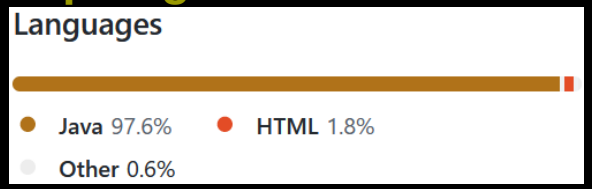
- TeaVM has a subproject, called Flavour. Flavour is a framework for writing single-paged web applications. It provides an HTML template engine with data binding, which is very similar to popular JavaScript frameworks like Angular, React, Vue.js and so forth. Additionally, Flavour provides facilities to generate and parse human-readable URLs, as well as binding them to the browser's address line via history API.

  Another purpose of Flavour is: due to several reasons it's hard to support entire JDK in TeaVM, especially things like reflection, class loading, resources, threads, thus most Java libraries for data serialization and network communication are unavailable in TeaVM. Flavour reimplements reasonable subsets of Jackson and JAX-RS client without a single line of reflection.

  Flavour *is not* a server-side framework. You are supposed to write your back-end code using "normal" JDK like OpenJDK, Oracle JDK using your favourite framework like Spring, Java EE, Vert.x, etc. Flavour is a client-side framework that you can use together with your back-end code. To make this easier, Flavour tries to be as familiar to Java developers as possible.

  Source: https://teavm.org/docs/intro/overview.html

- https://github.com/konsoletyper/teavm-flavour



- https://teavm.org/docs/flavour
- …

2.1.1.2 Wasm

- WebAssembly support is in experimental status. It may lack major features available in JavaScript backend. There's no documentation yet and you should do many things by hands (like embedding generated `wasm` file into your page, importing JavaScript objects, etc). Look at samples/benchmark module. You should first examine `pom.xml` file to learn how to build `wasm` file from Java. Then you may want to examine `index-teavm.html` and `index-teavm.js` to learn how to embed WebAssembly into your web page.

  Source: https://github.com/konsoletyper/teavm

- **https://github.com/fermyon/teavm-wasi**
  Friendly fork of TeaVM with support for WASI and the WebAssembly Component Model.
- **…**

X. Wrap-up

- **Polyglot programming** is more important now than ever before.

- Polyglot runtimes like **.Net, Wasm, GraalVM** are booming.

- **Cross-platform ecological integration** for Polyglot programming is the future.

Q & A

...

Reference

Slides/materials from many and varied sources:

- http://en.wikipedia.org/wiki/
- http://www.slideshare.net/
- https://blog.dotnetsafer.com/new-official-net-7-features-released-now-faster-and-lighter/
- https://nodogmablog.bryanhogan.net/2022/11/lambda-cold-starts-net-7-native-aot-vs-net-6-managed-runtime/
- https://news.ycombinator.com/item?id=31431387
- https://learn.microsoft.com/en-us/dotnet/fundamentals/implementations#applicable-standards
- https://www.graalvm.org/latest/reference-manual/polyglot-programming/
- https://www.wasm.builders/pauldotyu/exploring-net-webassembly-with-wasi-and-wasmtime-2a32
- https://www.fermyon.com/blog/dotnet-wasi
- https://visualstudiomagazine.com/articles/2022/02/18/net-7-webassembly.aspx
- https://dev.to/smurawski/getting-started-with-hippo-a-webassembly-paas-part-1-5470
- https://github.com/migueldeicaza/mono-wasm-mono

- https://github.com/unoplatform/uno
- https://learn.microsoft.com/en-us/aspnet/core/blazor/host-and-deploy/webassembly?view=aspnetcore-7.0
- https://github.com/kg/ilwasm
- https://github.com/sq/JSIL
- https://www.fermyon.com/blog/python-wagi
- https://www.infoworld.com/article/3613873/microsoft-gets-serious-about-webassembly.html
- https://www.codenameone.com/img/github/architecture.pdf
- https://www.fermyon.com/wasm-languages/java
- http://blog.dmitryalexandrov.net/webassembly-for-java-developers/
- https://news.ycombinator.com/item?id=25978053
- https://www.infoworld.com/article/3619608/14-hot-language-projects-riding-webassembly.html
- https://thenewstack.io/webassembly-5-predictions-for-2023/
- …