"Swift Runs Everywhere. I Checked."

Swift Runs Everywhere. I Checked.

"Swift runs everywhere" is usually a slide. A roadmap bullet, a conference promise, a footnote about a port someone started. I wanted to know what it looks like when it is not a slide, so I held my own projects to it: every platform claim must be a CI gate that runs on every push, and at least one build per project must be something you can click right now.

Four projects pass that bar today. A static site generator, a Markdown to PDF engine, a YAML parser, and a 2D vector graphics engine. All pure Swift. Between them they cover macOS, Linux, Windows, and the browser.

Exhibit one: the page you are reading

This site is built by TileDown, a static site generator written in Swift. The math on my posts is typeset to SVG at build time, the charts and diagrams are fenced text blocks rendered by the engine, and every article ships its own PDF from the same Markdown source. I wrote about the why in Why I built TileDown.

The part that belongs in this post is the CI table. On every push, TileDown builds and tests on macOS and on Linux in a Swift 6.1 container, the core targets build on Windows, and the math playground product cross-compiles to wasm32-unknown-wasip1 with the official Swift WebAssembly SDK. Four platforms, four green gates, one codebase. The generator that produced this page would produce it on a Linux box just as happily.

Exhibit two: a PDF engine in your browser tab

pdf.aleahim.com is MarkdownPDF compiled to WebAssembly. Type Markdown on the left, get a real PDF on the right, with no server involved. The engine parses the Markdown, lays out the document, and writes the PDF bytes directly in Swift, no LaTeX, no headless browser, no C library underneath. The wasm ships gzipped at about eighteen megabytes, downloads once, and is cached after that.

The same engine runs the full witness test suite on macOS and Linux, where the output PDFs are checked by independent tools including veraPDF for PDF/UA and PDF/A conformance. Windows and WebAssembly are build gates that compile the engine on every push. I covered the engine itself in Markdown to PDF in your browser, in pure Swift.

Exhibit three: a YAML parser with receipts

PureYAML is a dependency-free YAML package: no external SwiftPM dependencies, no bundled C sources, and no Foundation requirement in the library target. Its single CI workflow gates macOS, Linux, Windows, and WASI on every push.

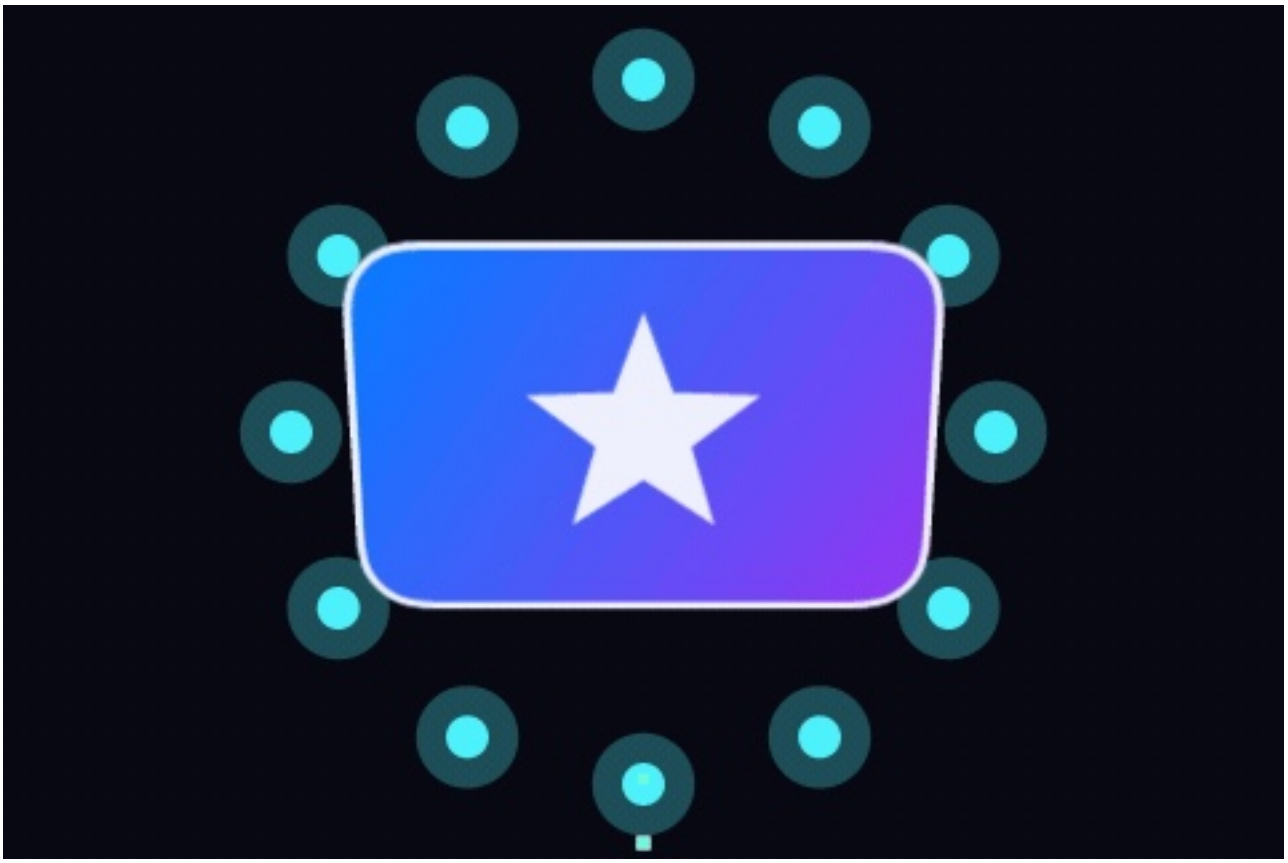
Correctness is measured, not asserted. A comparison harness runs PureYAML and Yams side by side over a corpus of 111 real-world documents, OpenAPI specs, Kubernetes manifests, GitHub Actions workflows. In the latest checked-in run both parsers accept the same 110 documents and reject the same intentionally damaged one, with zero disagreements between them.

And because a CI badge is less fun than a demo, there is a live streaming benchmark that fetches multi-megabyte YAML files and parses them as streams, in Swift, inside your browser tab, reporting fetch time, parse time, and throughput as it goes.

Exhibit four: a graphics engine that brings its own canvas

PureDraw is a dependency-free 2D vector graphics engine, a "Virtual PostScript Machine" with an API compatible with CoreGraphics and the HTML5 Canvas, written without either of them. Zero external dependencies, zero bundled C sources, no Foundation requirement in the core. The capability showcase at the top of its README, a bezier-path character, gradients, blend modes, shadows, a 3D cube with a projective grid texture, is a PDF rendered entirely by the engine itself, from the test suite.

This is the project where portability earns its keep most visibly: graphics is the domain everyone assumes belongs to the platform. PureDraw builds and runs its full test suite on macOS, on Linux in a Swift container, and on Windows, in both debug and release, and the library cross-compiles to WebAssembly on every push. The drawings that come out are the same bytes everywhere, because nothing underneath them belongs to any one platform.



And you do not have to take the CI's word for any of it. That badge is not a video. It is a few lines of PureComposition, a small language with a grammar of its own, compiled and rendered frame by frame. Glow Draw runs the same engines as WebAssembly right in your browser tab. PureDraw is CoreGraphics rebuilt from nothing, PureLayer is CoreAnimation rebuilt the same way, and PureComposition is the language that drives them: an EBNF grammar, a parser, a compiler that lowers it to a PureDraw virtual-machine program, and a command-line engine that runs the whole pipeline with no Mac in sight. The drawing API, the layer compositor, and the language on top, all reconstructed in portable Swift. If rebuilding Apple's graphics stack from first principles is the kind of problem you would want to talk about, get in touch.

If Swift is your language and you have been told it stops at the Mac, the CI logs say otherwise.