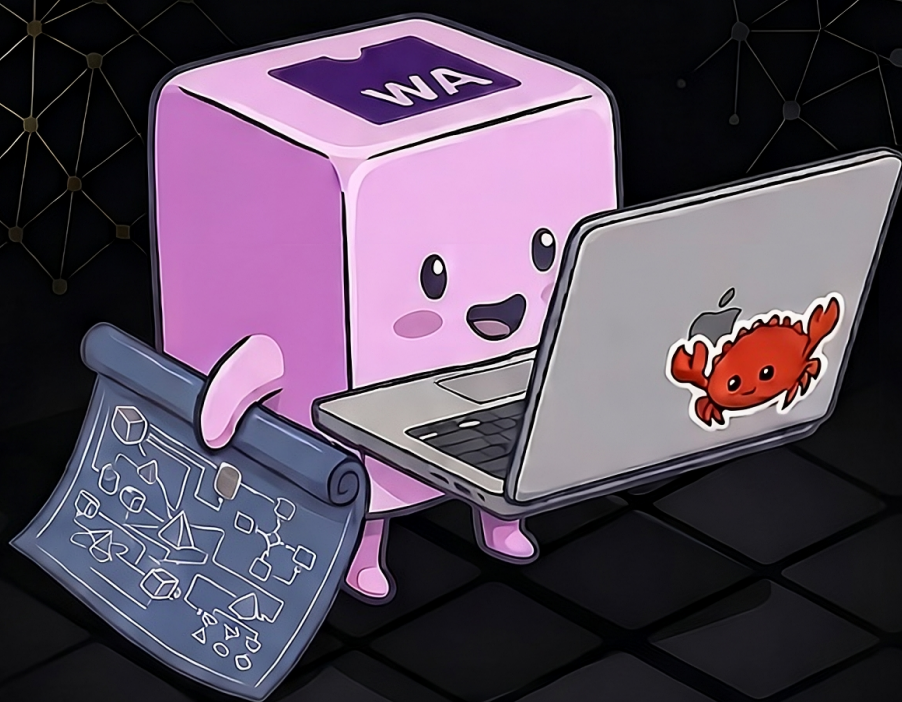


BUILDING SERVERLESS APPLICATIONS WITH WEBASSEMBLY & SPIN



By Thorsten Hans

Building Serverless Applications With WebAssembly & Spin

Thorsten Hans

This book is available at

<https://leanpub.com/building-serverless-apps-with-wasm-and-spin>

This version was published on 2026-10-13



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2026 Thorsten Hans

Contents

About the Author	i
Acknowledgments	ii
Foreword	iii
Foreword by Radu Matei	iii
Foreword by Matt Butcher	iii
Introduction	v
Who This Book Is For	v
Who This Book Is Not For	vi
What We Will Build	vi
How to Read This Book	vii
AI Disclosure, Production Notes, and Version History	viii
I: The Foundation	1
Chapter 1: The Evolution of Serverless & WebAssembly	2
Where Containers and Traditional FaaS Fall Short	3
What Modern Serverless Demands	4
WebAssembly Is the Foundation	5
Why Spin?	11
Why Rust?	11
Languages Beyond Rust	12
Chapter 2: What Is Spin?	15
The History of Spin	15
Spin 4 and the Modern Component Stack	15
Spin and the Capability-Based Sandbox	15
Spin and WebAssembly Components	16
Spin’s Trigger-Executor Pattern	17

Spin SDKs: Developer Ergonomics and Velocity	18
II: Setup & Spin Idioms	19
Chapter 3: Quickstart: Your First Spin Application	20
Setting Up Your Development Machine	21
Scaffolding Spin Applications	23
The Spin Workflow: Code, Build, Run	26
Chapter 4: Spin Application Idioms	30
The Spin Application Manifest	31
Configuration with Spin Variables	32
Using Environment Variables	33
Spin Trigger Types	34
The Spin Key-Value Store	36
III: Task-Engine: The End-to-End Application	39
Chapter 5: Building the Task-Engine	40
Scaffolding Multi-Component Applications	40
Task-Engine Architecture	41
Implementing the task-api Component	42
Building and Serving the Frontend	44
Integrating External APIs: Adding Notifications	45
Component Composition: The classifier	46
Implementing the Background Worker	48
Chapter 6: Operational Excellence	50
Structured Logging	51
Continuous Integration with GitHub Actions	53
Unit Testing Business Logic	58
Integration Testing Spin Apps	59
IV: Going Live	61
Chapter 7: Securing and Hardening Your Application	62
Hardening Spin Manifests	63
Hardening Application Code	66

CONTENTS

Signing and Verifying Spin Applications with <code>cosign</code>	66
Software Bill of Materials (SBOM)	68
Chapter 8: To the Global Edge with Akamai Functions	70
What Is Akamai Functions?	70
Constraints: The HTTP-Only Nature of the Edge	70
Task-Engine Deployment to Akamai Functions	71
Data at the Edge: State in Distributed Environments	72
Scheduling the Background Worker	73
Continuous Deployment with GitHub Actions	73
Chapter 9: Running WebAssembly on Kubernetes with SpinKube	75
The SpinKube Architecture	75
Cluster Setup and SpinKube Installation	75
Installing Application Dependencies	77
The <code>SpinApp</code> and <code>SpinAppExecutor</code> CRDs	77
Deploying the Task-Engine <code>SpinApp</code>	78
Scheduling the Background Worker	79
Continuous Deployment with GitHub Actions	79
 V: Make Spin Your Own	 81
Chapter 10: Extending Spin	82
Authoring and Distributing Custom Spin Templates	83
Authoring Functional Spin Plugins	84
Authoring Custom Trigger Plugins	85
Chapter 11: Contributing Upstream	88
Contributing Beyond Runtime Code	89
Navigating the Spin Ecosystem	89
Connecting on CNCF Slack	89
Participating in Community Meetings	89
Making Your First Contribution	89
 VI: Real-World Spin Apps	 90
Chapter 12: Practical Architectural Patterns	91
Pre-Initialization with Wizer	92
Sandboxed Agent Tools with MCP	92

LLM-Powered Chatbots	92
Dynamic Image Processing at the Edge	92
Anti-Scraping Through Edge HTML Rewriting	93
Edge API Aggregation and Fan-Out	93
 Wrapping Up	 95
Bringing It All Together	96

About the Author

I am a cloud-native software engineer and architect with a deep passion for building scalable, resilient systems. Over the course of my career, I've had the privilege of guiding engineering teams through the evolving landscape of cloud computing, from early containerization strategies to the third wave of compute: WebAssembly.

My journey with WebAssembly started with early exploration years ago, but I quickly put it aside because the tooling was too rough for practical, everyday use.

That all changed when I stumbled upon [Krustlet](https://krustlet.dev)¹, which allowed me to run WebAssembly workloads directly on Kubernetes. Through that ecosystem, I discovered Spin in May 2022, right around its v0.2.0 release. The developer experience clicked immediately. Just a month later, in June 2022, I introduced Spin to a developer audience for the first time at the CloudLand conference in Germany.

From that point on, I was hooked. I spent the next couple of years continuously producing content, writing articles, and delivering conference talks about Spin. That community advocacy eventually evolved into my day job: I officially joined Fermyon as a Senior Developer Advocate on February 1, 2024. As the WebAssembly ecosystem scaled, so did we, and on January 1, 2026, I joined Akamai as a Senior Developer Advocate following their [acquisition of Fermyon](https://tinyurl.com/akamai-acquired-fermyon)².

Today, I specialize in translating these bleeding-edge technologies into practical, production-ready tools. My work heavily revolves around Rust, Go, distributed systems, and the wider CNCF ecosystem.

When I am not writing Spin applications or poking pods on Kubernetes, you can usually find me exploring new trails on my mountain bike, playing soccer, or working out at the gym.

¹<https://krustlet.dev>

²<https://tinyurl.com/akamai-acquired-fermyon>

I would love to hear your thoughts on the book or see what you build with Spin. You can connect with me on [LinkedIn](#)³, follow my open-source contributions on [GitHub](#)⁴, and read my [blog](#)⁵.

Acknowledgments

Writing a technical book on a rapidly evolving ecosystem is never a solo endeavor, and this manuscript is the result of immense support from the community. First, my deepest thanks to Radu Matei and Matt Butcher for their early support, their vision for WebAssembly, and for graciously writing the forewords.

To the entire Spin and upstream WebAssembly and WASI community: thank you for building a framework that takes the academic promise of WebAssembly and makes it genuinely fun to work with.

To my technical reviewers (Christian Wenz, Karthik Ganeshram, Jörg Neumann, and Ruckus Voxi), your sharp eyes, precise feedback, and candor saved me from countless inaccuracies, making this a fundamentally better book. Any remaining errors are entirely my own.

Finally, a massive thank you to my family. To Nina, Lena, Anna, and Emily, your patience during the countless evenings and weekends I spent buried in code, debugging HTTP triggers, and rewriting chapter drafts means more to me than I can adequately express.

This book exists because of your support.

³<https://linkedin.com/in/ThorstenHans>

⁴<https://github.com/ThorstenHans>

⁵<https://thorsten-hans.com>

Foreword

Foreword by Radu Matei

Thorsten has been part of Spin since its early pre-release days. He has built more Spin applications than anyone I know, contributed to the core, and spent years helping teams take their applications into production. Within the project, he has used that experience to advocate for developers and helped shape Spin into what it is today. That is why I am delighted he is the author of this book.

When we started Spin, we believed Wasm was the right unit of execution and deployment for serverless functions. Our goal was to abstract the underlying machinery and make building, testing, and deploying applications seamless. This ambition has guided much of our work on Spin, and I cannot think of a better guide to putting it into practice than this book.

My hope is that this book gives you the understanding and confidence to make Spin useful in your own work. Your functions, ideas, and experience will help the project grow in ways we could never have planned.

– [Radu Matei](#)¹ Co-creator of Spin · Chief Product Architect at Akamai

Foreword by Matt Butcher

First there were virtual machines. Then came OCI containers. But what if there was another cloud runtime—One optimized for fast startups and short execution times? That was the core idea that led us to create Spin.

My team at Microsoft was responsible for building Cloud Native tooling, mostly targeting Kubernetes. We had already churned out projects like Helm, Draft, Brigade, and OAM – all solidly rooted in Kubernetes. We had recently turned our attention to Functions as a Service (FaaS), and that in turn led us to

¹<https://www.linkedin.com/in/mateiradu>

WebAssembly. Its secure sandboxing technology, broad language support, and fast cold-start time made it an ideal candidate for a cloud runtime specific to FaaS.

When we left Microsoft to start Fermyon, our core vision was that we could build a “next wave of cloud computing” powered by WebAssembly. The very first day we were a functioning company, we started work on a new project: Spin. Our goal was simple: Make it so easy to build WebAssembly-powered function-style apps that a developer could go from blinking cursor to deployed app in two minutes or less.

Spin was open source from its inception. And I first met Thorsten Hans because of his early involvement in the Spin open source ecosystem. From my first conversation with him, it was obvious to me that he not only understood the merits of Spin and WebAssembly, but that his own background as an architecturally minded developer meant he was already thinking about (and building) Spin apps at a level of complexity far beyond what we ourselves had attempted. Thorsten joined our team and dedicated himself to defining the best practices for Spin application building. Now, a few years later, I can confidently say that Thorsten is the foremost expert on using Spin. He has contributed to the core of the system. He’s built probably hundreds of Spin apps. He’s taught workshops, spoken at conferences, and written documentation. This book is a distillation of Thorsten’s expertise.

The idea of Spin germinated at Microsoft, grew to fruition at Fermyon, has flourished as an open source project in CNCF. The open source SpinKube project enables many organizations to run Spin applications on a large scale within their own Kubernetes clusters. When Akamai acquired Fermyon, Spin became the foundation for Akamai Functions. As AI has emerged as both a source of new application patterns and a tool for generating code, Spin is showing its strengths in new ways. This book, borne from Thorsten’s knowledge and experience, will inspire you to create apps for a variety of environments and use cases. I believe you will find it as enjoyable and informative as I have.

– [Matt Butcher](#)² Co-creator of Spin · Vice President of Product Management at Akamai

²<https://www.linkedin.com/in/mattbutcher>

Introduction

Let me start with a confession:

I have deployed a lot of containers in my career, and for most of that time, I never questioned the model. A container was simply *the* unit of deployment. It's only when you put something radically lighter next to it that you realize how much you were paying – in cold-start latency, in image size, and in the sheer amount of operating system and dependencies you drag along just to run a few hundred lines of business logic.

This book is about that lighter thing.

We're going to build real, production-shaped serverless applications on top of **WebAssembly**¹ using **Spin**², the open-source framework that (in my very biased opinion) makes server-side WebAssembly genuinely pleasant to work with.

And we'll do it in **Rust**³, because Rust and WebAssembly grew up together, and the developer experience shows it.

Who This Book Is For

This book is for you if you're a software engineer, architect, or tech lead. You know your way around HTTP, package managers, and the terminal.

You may have written Rust before, or you may just be curious about Rust; either is fine. I'll explain the Spin-specific and WebAssembly-specific parts, and I'll trust you on the rest.

What you **do not** need is prior WebAssembly experience. We start from first principles, and we lead with the “Why” before the “How.”

¹<https://webassembly.org>

²<https://spinframework.dev>

³<https://rustlang.org>

Who This Book Is Not For

This is not an introductory book on Rust.

While you don't need to be an expert in Rust, we will move quickly. The code examples rely on standard Rust idioms, patterns, and ecosystem crates. We will dive deep into Spin-, Wasm-, and Component Model-specific patterns, but we won't stop to explain basic Rust syntax, ownership, error handling with `Result`, or `async` traits.

If you are completely new to Rust, I recommend spending a weekend (or two) with [The Rust Programming Language book](https://doc.rust-lang.org/book)⁴ first to get comfortable with the syntax. Once you can read basic Rust code, come back right here.

What We Will Build

Rather than a scattering of disconnected “Hello, World” examples, we build one sample application across the whole book and grow it chapter by chapter. I call it the **Task-Engine** – a small but functional task-management app. By the end, it consists of several independent, yet interconnected, WebAssembly components.

It grows real capabilities along the way:

- **Persistence** that uses the exact same code across different deployment targets.
- **Outbound Networking** with precise security controls in place.
- **Background Worker** to perform work on a schedule.
- **WebAssembly Component Composition** to encapsulate business logic and build reusable components.

Once we have the core features of Task-Engine implemented, we'll dive into topics like *automation* and *security hardening*. From there, I'll assist you in *Going Live* with the application by deploying it to two very different production targets (out to the global edge with **Akamai Functions** and onto Kubernetes

⁴<https://doc.rust-lang.org/book>

with **SpinKube**) without rewriting a single line of application code or recompiling the application.

That portability is not a marketing slide; it's the whole point, and you'll feel it firsthand.

Finally, we explore how to **Make Spin Your Own** by authoring custom templates, developing CLI plugins, building custom trigger executors, and contributing to the open-source community.

How to Read This Book

It's meant to be read front-to-back the first time; later chapters lean on earlier ones because we're building one app, not twelve. Throughout, you'll find short asides that look like this:



These are the things I'd lean over your shoulder and tell you if we were pairing. Sometimes a shortcut, sometimes a landmine I've already stepped on so you don't have to.



Crucial points, or pitfalls that might bite you in the long run, are highlighted using this prominent warning sign.



Last but not least, these highlights represent me taking care of your caffeine levels.

AI Disclosure, Production Notes, and Version History

This book is written in [Markua](https://markua.com)⁵ and the digital version is published via [Leanpub](https://leanpub.com)⁶. The code examples, configurations, and architecture diagrams are verified against the target environment below.

- **Publication Date:** October 2026
- **Target Stack:** Spin v4.1.0, Rust 1.98+, Spin SDK 7.0+, WASI Preview 3 (targeting wasm32-wasip2)
- **Companion Code Repository:** [thorstenhans/task-engine](https://github.com/thorstenhans/task-engine)⁷

AI Disclosure

I want to be fully transparent about how I used generative AI while creating this book. The **entire manuscript was written by hand**, without any generative AI or agentic tools. That might sound unusual these days, but it is the honest truth.

I used AI (Google Gemini Pro) for basic proofreading before sending out review copies to ensure a smooth reading experience for everyone. Additionally, **I used AI (Google Gemini Pro) to generate the book's mascot and section illustrations**, as creating visually appealing art is simply not a skill of mine.

All technical illustrations and diagrams were created by me **without AI involvement** using [Figma](https://figma.com)⁸.

⁵<https://markua.com>

⁶<https://leanpub.com>

⁷<https://github.com/thorstenhans/task-engine>

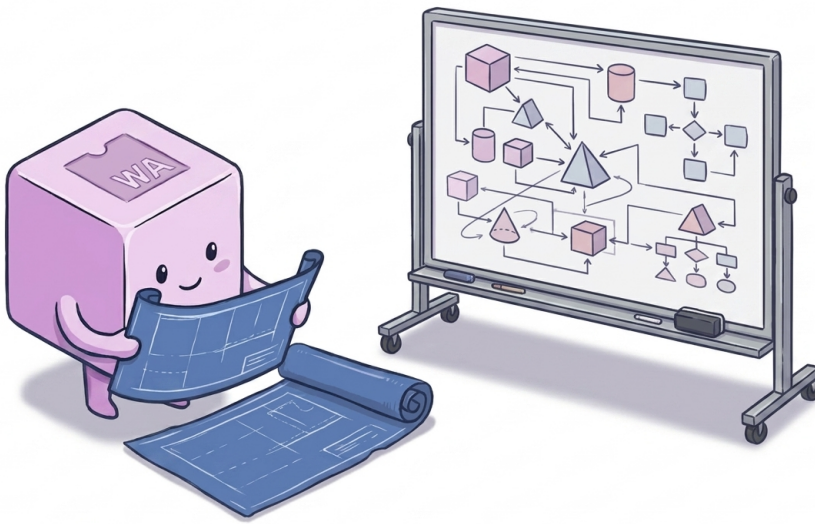
⁸<https://figma.com>

Version History and Changelog

- 0.1.0 (2026-10-13) - 1st Public Release
 - Incorporated review feedback
 - 0.0.10 (2026-10-10) - 3rd Review Delivery
 - Incorporated review feedback
 - Updated illustrations and screenshots
 - 0.0.9 (2026-10-07) - 2nd Review Delivery
 - Hand out full manuscript to trusted reviewers
 - 0.0.8 (2026-09-11) - 1st Review Delivery
 - Hand out subset of the manuscript to trusted reviewers
-

Let's get into *why* any of this matters before we write a line of code.

The Foundation



The Evolution of Serverless & WebAssembly



Grab a coffee now! This chapter is all “why” and no code, so it’s the perfect place to enjoy it before it goes cold.

Every few years, our industry redraws the line between “your code” and “the machine it runs on.” Physical servers gave way to virtual machines, virtual machines gave way to containers, containers gave us Kubernetes and, eventually, serverless functions. Each step made the unit of deployment smaller, faster to start, and easier to reason about.

WebAssembly is the next step on that line, and this chapter is about *why* it exists at all.



Figure 1.1. The Compute Continuum

We won’t touch the Spin CLI or its SDKs yet. Before we can appreciate what Spin does for us, we need to be clear about the problems it’s solving and where it helps us. So we’ll do three things:

1. Look at where **containers** and **traditional Functions-as-a-Service (FaaS)** hurt.

2. Introduce **WebAssembly** as a genuinely new option: not a replacement for everything, but as the third wave of cloud computing.
3. Explain why **Spin** is the framework I reach for when I want to build serverless apps, and why **Rust** is such a natural fit for it.

By the end of this chapter, you'll understand the problem domain well enough that every design decision in Spin will feel obvious rather than arbitrary.

That's the goal.

Where Containers and Traditional FaaS Fall Short

Let me be clear up front: **containers are wonderful**, and I'm not here to bury them. They solved a real problem ("It works on my machine"), and they did it well enough to reshape the entire industry. However, every tool has a grain. When you push containers in the direction of *fine-grained, bursty, event-driven compute*, you start rubbing against that grain. Hard.

Containers Carry a Full OS Userland

A container image is your code plus the userland: a base OS layer, system libraries, a language runtime, maybe even a package manager you forgot to strip out. Even a "small" image is tens to hundreds of megabytes; a careless one could be gigabytes. That weight shows up everywhere you'd rather it didn't:

- **Cold starts.** Scheduling a pod, pulling an image, starting a runtime – that's seconds, not milliseconds, in the unlucky case. If you scale to zero to save money, your users pay for it in latency on the next request.
- **Density.** Each container is a process (or several) with its own namespaces and cgroups. You can pack a lot onto a node, but "a lot" is maybe hundreds, not thousands.
- **The security boundary is the whole kernel.** A container shares the host kernel and, by default, has broad reach *inside* its sandbox. Locking it down is a configuration exercise (seccomp, AppArmor, dropped capabilities) that you have to remember to do, correctly, every time.

None of this is fatal for a long-running service that starts once and keeps serving traffic for weeks. It's death by a thousand cuts for a function that should spin up, handle one request, and vanish.

Traditional FaaS Trades One Problem for Another

Functions-as-a-Service (FaaS), like classic cloud “serverless,” tried to fix the ergonomics. Write a handler, forget the server, scale automatically. Lovely. But under the hood, most FaaS platforms are still provisioning a container or a micro-VM per function, and that means:

- **Cold-start latency** is a first-class, well-documented pain. A JVM-based function pulling in a fat dependency tree can take a full second (or more) to answer its first request. Node.js does better, but it’s still hundreds of milliseconds in the general case.
- **Vendor lock-in** creeps in through the handler signature, the packaging format, and the proprietary event shapes. Your “portable function” is portable right up until you try to move it to a different FaaS platform or cloud vendor.
- **Idle cost and keep-warm hacks.** Because cold starts hurt, teams keep functions warm with synthetic pings, which quietly defeats the scale-to-zero economics that made FaaS attractive in the first place. Cloud vendors provide premium tiers, which keep idle resources allocated to “hide” the cold-start penalty.

Notice the pattern. Containers are too heavy to start instantly; traditional serverless is often built on containers or micro-VMs, so it inherits the weight and papers over it with automation. We keep optimizing the same fundamental unit instead of questioning it.

What Modern Serverless Demands

If we describe the ideal unit of serverless compute from scratch, the wishlist writes itself:

- Starts in **milliseconds**, so scale-to-zero has no latency penalty
- Results in a **tiny package**, so distribution and cold-pull are cheap
- **Resource-efficient**, so a single host can run thousands of distinct workloads
- **Strictly sandboxed**, so running untrusted code is safe without a checklist

- **Portable:** The same artifact can run everywhere: any CPU, any OS, any cloud, no rewrite

That's not a description of a container. It's a description of a **WebAssembly component** – which is exactly where we're heading.

WebAssembly Is the Foundation

So what is WebAssembly? The specification gives us a delightfully unglamorous one-liner, and I want to anchor everything on it:

WebAssembly (abbreviated Wasm) is a binary instruction format for a stack-based virtual machine.

Source: webassembly.org¹

That sentence is doing a lot of work, so let's unpack the three ideas hiding in it, because each one maps directly to a value we care about on the server and in the cloud.

Binary Instruction Format

Wasm is compiled *bytecode*, not source and not a language you write by hand (although you can write it by hand using WAT). You write Rust (or Go, or C, or...); a compiler emits a compact `.wasm` binary.

“Compact” matters: the format was explicitly designed to be small over the wire and fast to decode. This is where the tiny artifact and fast cold start properties begin. To put the real-world impact of this design into perspective, consider how WebAssembly compares to traditional deployments across two critical operational metrics:

Artifact Size: Because WebAssembly components package only your application code and its direct dependencies (stripping away the entire operating system userland, background utilities, and filesystem found in a standard Docker image), the resulting binaries are typically measured in mere megabytes, or even kilobytes.

¹<https://webassembly.org>

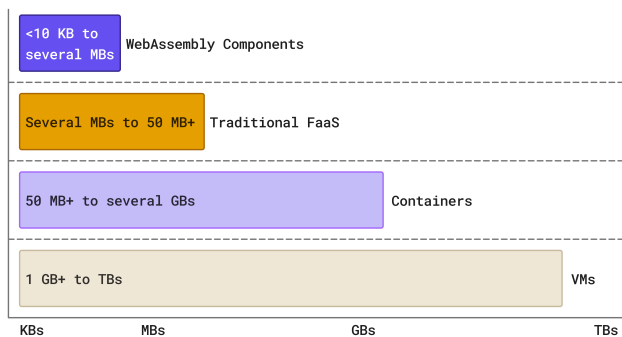


Figure 1.2. Comparison of Typical Distribution Sizes

Cold Start Execution: Without a guest OS to boot or a heavy container network stack to initialize, a runtime like Spin can load, decode, and execute a WebAssembly component in under a millisecond. This near-instant initialization effectively eliminates the notorious cold start penalty also associated with traditional FaaS.

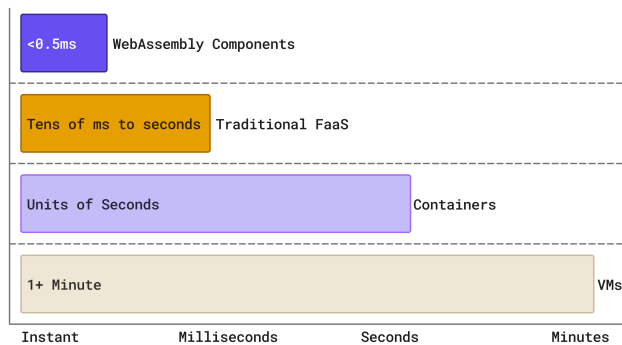


Figure 1.3. Comparison of Cold-Start Times

Stack-Based Virtual Machine

The code runs on an abstract stack machine with a defined instruction set, independent of any real CPU. Your amd64 laptop and an arm64 edge node run

the *same bytes*. A runtime like [Wasmtime](https://wasmtime.dev)² compiles that bytecode to native machine code (ahead-of-time or just-in-time), so you get near-native speed without shipping a native binary per architecture.

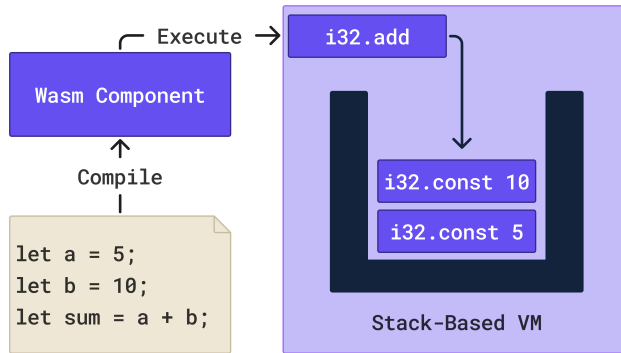


Figure 1.4. Stack-based Virtual Machine

Here's the part experienced engineers appreciate: before a single instruction executes, the runtime **validates** the module. It type-checks it and proves that memory accesses and control flow stay in bounds. A malformed or unsafe module is **rejected at load time**, not caught later with a segfault. Illegal operations at runtime don't corrupt anything either; they **trap**, aborting cleanly and reporting the failure to the host. No undefined behavior leaks into your process.

²<https://wasmtime.dev>

Sandboxed by Default

A Wasm module's entire world is a single, bounds-checked block of **linear memory**. It cannot address the host's memory, another module's memory, or anything else!

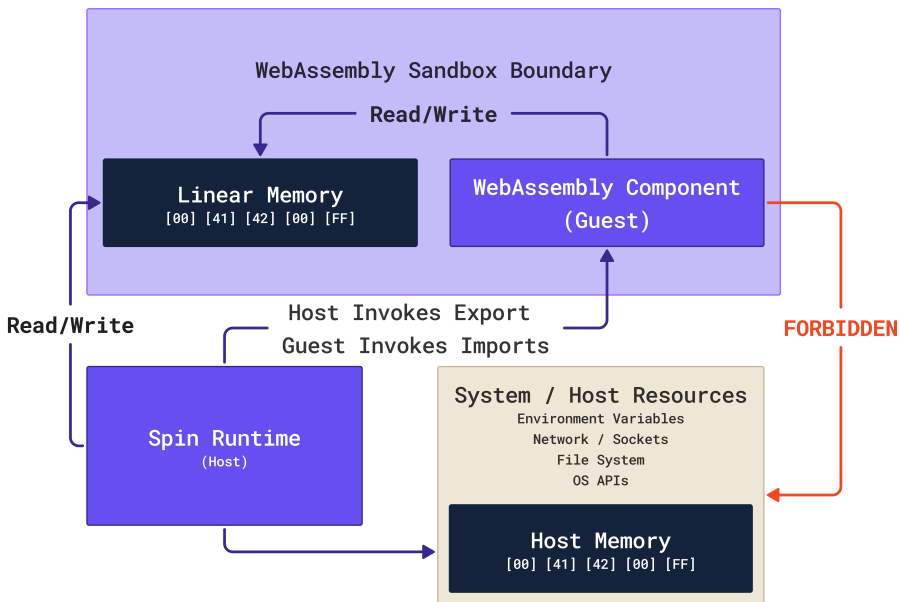


Figure 1.5. WebAssembly Sandbox

And, crucially, a bare module **cannot** touch files, the network, the clock, or environment variables *at all*. It can crunch numbers in its sandbox and nothing more, unless the host hands it a capability. That's WASI³, and it takes center stage now.

WASI: Capabilities, Not System Calls

That complete isolation is great for security, but a microservice or serverless workload that can't read a file or make an HTTP request is just a very secure

³<https://wasi.dev>

desk toy. This is exactly where WASI (the WebAssembly System Interface) comes in. Think of WASI as a standardized set of system calls designed specifically for the WebAssembly sandbox. Instead of letting a module talk directly to the operating system, WASI defines a capability-based contract. The host environment acts as a strict gatekeeper: if your WebAssembly application needs to read a config file or open a network socket, the host must explicitly grant that specific capability to the module before execution.

Crucially, WASI gives us POSIX-like system capabilities without tying your code to a specific operating system or architecture. A WebAssembly module compiled against WASI doesn't care whether it is running on macOS, Linux, or Windows. The interface to files, clocks, and sockets remains identical. With WASI Preview 2 (and the WASI 0.3 ecosystem), this model extends beyond bare system primitives to rich, typed interfaces defined via WebAssembly Interface Types (WIT), allowing modules to **import** and **export** high-level capabilities cleanly and safely.

Cross-Platform Portability

Because WebAssembly executes within the stack-based virtual machine, it achieves true portability across hardware, cpu architectures and operating systems. For us as developers, the real magic is what this architecture eliminates from our daily workflow: the dreaded multi-arch build matrix.

If you've ever wrestled with docker buildx to deploy a containerized app across Apple Silicon laptops, amd64 cloud clusters, and arm64 edge nodes, you know the pain. WebAssembly completely sidesteps this operational overhead. You compile your application into a single `.wasm` module exactly once. By leveraging the binary instruction format and WASI to abstract away the underlying CPU and OS, that exact same artifact executes flawlessly anywhere a WebAssembly runtime exists. No cross-compilation configs, no redundant CI pipelines, and zero container bloat.

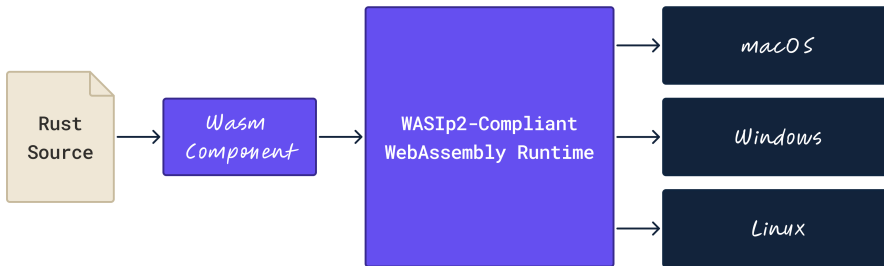


Figure 1.6. WebAssembly Portability

The WebAssembly Sweet Spot

Put those properties together, and you get a unit of compute that is genuinely different from what came before – not strictly better, *different*, with its own sweet spot:

- **Cold start in milliseconds:** No OS to boot, no image to pull, and no language runtime to warm. A precompiled module is essentially memory-map-and-go. Scale-to-zero stops being a latency tax.
- **Ultra-dense:** An instance is a memory arena plus some bookkeeping, not a process with its own userland. This allows a host to pack far more of them.
- **Portability:** Compile once, run anywhere a runtime exists. It's the JVM's old promise, but at the bytecode layer, language-agnostic, and with a much smaller trusted computing base.
- **Secure by default:** Deny-by-default. Ideal for multi-tenant platforms and for running third-party code you don't fully trust.



You may hear “Wasm is 1000× faster than containers.” Don't repeat it. The *microsecond* figure is instantiation time, not end-to-end throughput. For long-running, CPU-bound work, a well-tuned container can match or beat Wasm in a steady state. Be precise, and you'll be believed! Wasm wins decisively where **startup speed and density** dominate: short-lived, bursty, event-driven, multi-tenant. That's a big part of a typical serverless architecture, but it does not encompass **everything**, and saying so builds trust.

The honest framing – the one I'd stake my reputation on – is this. In 2019, Docker's co-founder Solomon Hykes said the following on X (previously Twitter):

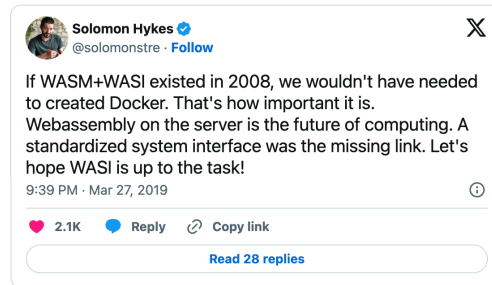


Figure 1.7. Source: Solomon Hykes on X

He wasn't predicting Docker's death. He was pointing at the same thing I am: for a large and growing class of workloads, there's now a lighter, safer, more portable unit than a container.

WebAssembly lays the foundation for the third wave of cloud computing, standing next to VMs and containers, and Spin is how we build on it productively.

Why Spin?

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

What Spin Provides

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Why Spin Stands Out

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Why Rust?

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The Synergy Between Rust and WebAssembly

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Anatomy of a Spin Component

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

When to Choose Rust

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Languages Beyond Rust

Spin is **language-agnostic by design**, allowing you to use any language that can be compiled into a WebAssembly component. At the time of writing this book, Spin provides first-class support for:

- Rust
- Go
- TypeScript / JavaScript
- Python

Note: The list of supported languages keeps growing as more and more programming languages provide first-class support for WebAssembly, WASI, and the WebAssembly Component Model.

How Interpreted Languages Compile to WebAssembly

You may wonder how Spin turns interpreted languages such as JavaScript or Python into a WebAssembly component.

In the case of JavaScript, your application relies on a specialized, optimized JS runtime called [StarlingMonkey](https://github.com/bytecodealliance/StarlingMonkey)⁴. StarlingMonkey is based on [SpiderMonkey](https://spidermonkey.dev)⁵. It is designed to perform as efficiently as possible within the bounds of a Wasm component.

At compile time, StarlingMonkey, your source code, and all its dependencies are loaded. The global scope is evaluated. At this point, a snapshot is compiled to Wasm and written to disk, resulting in an application that is pre-warmed and ready to serve incoming requests immediately. As a result, JavaScript apps compiled into Wasm components **start faster** than regular JavaScript.

Polyglot Component Composition

Rust it is for this book, but it doesn't have to be Rust only!

A Spin application can consist of multiple components. For each component, you can choose a different programming language if you want to, or if a particular language helps you achieve the desired goal in less time or with less code.

Additionally, WebAssembly component composition allows you to stitch together multiple components, regardless of which language was used to implement those building blocks. Instead of juggling multiple languages across components in our sample app, however, I want to teach all necessary concepts *clearly* and show *idiomatic* code. Rust lets me do that with the least friction and the most honesty about what's really happening.

The stage is set. We've explored why containers and traditional serverless architectures struggle under modern latency and resource constraints, how

⁴<https://github.com/bytecodealliance/StarlingMonkey>

⁵<https://spidermonkey.dev>

WebAssembly changes that equation from first principles, and why pairing Spin with Rust gives us the foundation to build something radically lighter.

Now that the “why” is clear, it’s time to examine the “how.” In the next chapter, we’ll open the hood and explore the core architecture and mechanics that make Spin tick.

What Is Spin?

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The History of Spin

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Spin 4 and the Modern Component Stack

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Spin and the Capability-Based Sandbox

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Ambient Authority and Its Absence

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Granting Capabilities in Spin

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Two Critical Caveats

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Spin and WebAssembly Components

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The Problem with Plain WebAssembly Modules

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Interfaces Described in WIT

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Shared-Nothing Composition

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Mental Models for Components

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Why Component Composition Matters

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Spin's Trigger-Executor Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Built-In Triggers vs. Plugin Triggers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Under the Hood: `wasi:http`

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The Spin 4.0 Concurrency Model

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

What It Means for Your Rust Code

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Spin SDKs: Developer Ergonomics and Velocity

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The Rust Experience: Minimal Ceremony, Maximal Ergonomics

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Setup & Spin Idioms

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Quickstart: Your First Spin Application

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Setting Up Your Development Machine

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Install Prerequisites

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Install the Spin CLI

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Install Spin with mise

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Install Spin with Homebrew

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Verify Your Spin Installation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Install Rust

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Update the Rust Toolchain

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Install the Rust Target

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Install Spin Templates

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Run a Smoke Test

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Scaffolding Spin Applications

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Running `spin new`

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Generated Project Structure

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

spin.toml: The Application Manifest

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Cargo.toml: Library Crate Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

.gitignore: Version Control Defaults

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The Spin Workflow: Code, Build, Run

Time to write some Rust and proactively run through the *Spin Workflow*. We'll replace the generated `src/lib.rs` with a clean Spin 4.0-style HTTP service, build it, run it, and hit it with `curl`. Small scope, complete result.

To do so, we'll use the `hello-rust` application we scaffolded in the previous section.

Code

Open `src/lib.rs` and replace its contents entirely:

```
use spin_sdk::http::{IntoResponse, Request, StatusCode};
use spin_sdk::http_service;

#[http_service]
async fn handle_hello_rust(req: Request)
-> anyhow::Result<impl IntoResponse> {
    println!("Handling {:?} request to {:?}", req.method(), req.uri());
    if req.method() == "POST" {
        return Ok((
            StatusCode::METHOD_NOT_ALLOWED,
            "Sorry - No POST allowed!".to_string(),
        ).into_response());
    }
    Ok((StatusCode::OK, "Hello, from Rust!".to_string()).into_response())
}
```

Let's read it top to bottom, because every line earns its place.

`Request` and `StatusCode` are re-exports of the types from the standard `http` Rust crate. If you've used `hyper`, `axum`, or `reqwest` before, they'll feel familiar – Spin didn't invent bespoke HTTP types; it leaned on the ecosystem's.

The `#[http_service]` macro is the 4.0 way to declare an HTTP handler. It wraps your `async fn` and exports it as the component's entry point, so the Spin host can call it when the HTTP trigger fires. You write plain application code; the macro handles the component-model glue. That the function is

`async` is the headline change in Spin 4.0. The host APIs (outbound HTTP, key-value, variables, databases) are all `async` now, so your handler is `async` too, and you `.await` them naturally. Our `handle_hello_rust` handler doesn't `await` anything yet, but the shape is ready for when it does.

Returning `anyhow::Result<impl IntoResponse>` means you can use the question mark operator (?) throughout and let errors bubble up as a clean 500 status instead of writing match arms for every fallible call. It's the idiomatic error story you'll see in every handler in this book.

Build

No matter which language you use to build Spin apps, you'll always turn source code into WebAssembly components using the language-agnostic `spin build` command:

```
# Compile source down to WebAssembly
spin build
# Building hello-rust with `cargo build --target wasm32-wasip2 --release`
# ...
# Finished `release` profile [optimized] target(s) in 12.15s
# Finished building all Spin components
```

Because our application is written in Rust, `spin build` runs `cargo build --target wasm32-wasip2 --release` under the hood and emits a WebAssembly component at `target/wasm32-wasip2/release/hello_rust.wasm`. Since we already compiled this Spin app in the previous section, you may notice that subsequent builds are much faster. This is simply because dependencies haven't changed and don't need to be recompiled.

Run

```
# Launch the application
spin up

# Logging component stdio to ".spin/logs/"
# Serving http://127.0.0.1:3000
# Available Routes:
# hello-rust: http://127.0.0.1:3000 (wildcard)
```

Spin reads the application manifest (`spin.toml`), loads your component, wires it to the HTTP trigger, and prints a local address, by default `http://127.0.0.1:3000`. Leave it running and, in another terminal, send it a request:

```
# Send a GET request to the Spin App
curl -iX GET localhost:3000/foo

# HTTP/1.1 200 OK
# transfer-encoding: chunked
# date: Mon, 31 Aug 2026 10:57:30 GMT

# Hello, from Rust!
```

There it is – a request served by Rust compiled to a sandboxed WebAssembly component, cold-starting in microseconds. Back in the `spin up` terminal, you'll also see the Handling GET request to `http://localhost:3000/` line from our `println!` invocation, which is the crudest possible logging and completely fine for now (we get serious about observability in Part III).



You can save a few keystrokes and simply run `spin b`, which is a convenient alias for `spin build`. You can also combine compiling and running with a single `spin b --up` or `spin up --build`.

With that, you've made it through the fundamental Spin Workflow, summarized in the illustration below:

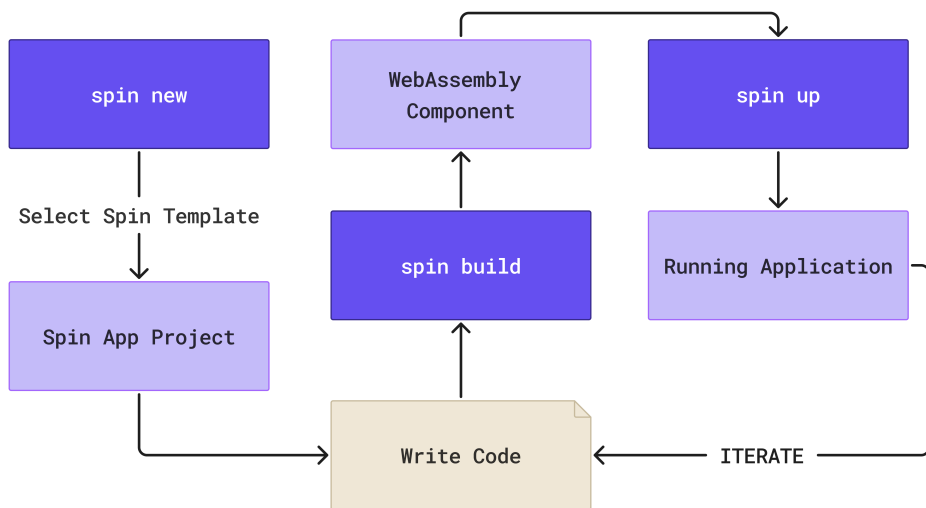


Figure 3.1. The Spin Workflow

That's your first success – your development machine is configured correctly, you've created your first application in Rust, walked through the *Spin Workflow*, and received an actual response by invoking the application with `curl`. No magic left unexplained.

Next, we'll slow down a bit and dive into some essential Spin idioms.

Spin Application Idioms

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The Spin Application Manifest

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Manifest Version and Core Metadata

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Triggers and Components

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

HTTP Routing Wildcards

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The Component build Table

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Capabilities: Deny by Default

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Build Profiles

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Component Dependencies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Configuration with Spin Variables

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Declaring Variables

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Mapping Variables to Components

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Reading Spin Variables from Rust

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Variable Providers and Precedence

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Using Environment Variables

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Defining and Using Environment Variables

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Environment Variables vs. Spin Variables

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Spin Trigger Types

HTTP is the trigger we've used so far, but it isn't the only one Spin offers. Before we build the Task-Engine, it's worth taking a quick tour of available Spin trigger types so you know they exist and understand, concretely, when you might want to use them, or not.

Non-HTTP Triggers

Spin's trigger types share the same model (a portable WebAssembly component with a deny-by-default capability grant) and differ only in *what invokes the component*. HTTP ships in the core runtime; the rest are either built in (Redis) or installed as trigger plugins.

Here is a quick look at popular trigger types other than HTTP:

The Redis Trigger

The Redis trigger invokes a component every time a message is published to a Redis (or [Valkey](https://valkey.io)¹) pub/sub channel. It's the classic event-driven pattern: a producer publishes a "task created" message, and a subscriber component reacts off the request path. The `[[trigger.redis]]` table binds a channel to a component:

```
[[trigger.redis]]
address = "redis://localhost:6379"
channel = "messages"
component = "subscribe"
```

It's a genuinely good fit *when you already run Redis or Valkey* and want event fan-out. You can find full details in the [Redis trigger documentation](https://spinframework.dev/v4/redis-trigger)².

¹<https://valkey.io>

²<https://spinframework.dev/v4/redis-trigger>

The SQS Trigger

The SQS trigger invokes a component when a message arrives on an AWS SQS queue: the same event-driven idea, but backed by a managed cloud queue instead of your own Redis. The trigger long-polls the queue and hands each message to the component:

```
[[trigger.sqs]]
queue_url = "http://localhost:9324/000000000000/test-queue"
max_messages = 1
idle_wait_seconds = 3
system_attributes = ["All"]
message_attributes = ["glonk"]
component = "sqs-consumer"
```

It's a plugin; source and details live in [spinframework/spin-trigger-plugins](https://github.com/spinframework/spin-trigger-plugins)³ (crate trigger-sqs).

The MQTT Trigger

The MQTT trigger connects to an MQTT broker and invokes a component when a message lands on a subscribed topic: the go-to for IoT and device-telemetry workloads. The broker connection is declared once; each `[[trigger.mqtt]]` subscribes a component to a topic (MQTT wildcards like `+` are allowed):

```
[application.trigger.mqtt]
address = "mqtt://localhost:1883"
username = "admin"
password = "password"
keep_alive_interval = "30"
```

```
[[trigger.mqtt]]
id = "telemetry-trigger"
component = "telemetry-handler"
topic = "sensors/+"
qos = "1"
```

³<https://github.com/spinframework/spin-trigger-plugins>

It is also a plugin; see [spinframework/spin-trigger-plugins](https://github.com/spinframework/spin-trigger-plugins)⁴ (crate `trigger-mqtt`).

The Command Trigger

The command trigger runs a component once, to completion, and then exits – no server, no port, no long-lived process. It’s the closest thing Spin has to a plain CLI program (it executes the `wasi:cli run export`), which makes it the natural home for one-shot jobs: a migration, a backfill, or a seeder. It’s also the recommended shape for a Kubernetes Job under SpinKube, where a cron trigger has nothing to schedule against.

```
[[trigger.command]]
component = "seed-data"
```

Plugin source: [spinframework/spin-trigger-plugins](https://github.com/spinframework/spin-trigger-plugins)⁵ (crate `trigger-command`).

The Cron Trigger

The cron trigger invokes a component on a schedule (“every five minutes” or “at midnight”) with no external request at all, driven by Spin’s own scheduler:

```
[[trigger.cron]]
component = "nightly-sweep"
cron_expression = "0 */5 * * * *" # Every 5 minutes
```

Plugin source: [spinframework/spin-trigger-plugins](https://github.com/spinframework/spin-trigger-plugins)⁶ (crate `trigger-cron`).

So we have purpose-built command and cron triggers, plus three event-driven triggers (Redis, SQS, and MQTT) that could each carry “react to something that happened.” For the Task-Engine, our end-to-end sample application, HTTP wins on the one axis we care about most: **Portability**.

Next, we dive into actually building Task-Engine.

⁴<https://github.com/spinframework/spin-trigger-plugins>

⁵<https://github.com/spinframework/spin-trigger-plugins>

⁶<https://github.com/spinframework/spin-trigger-plugins>

The Spin Key-Value Store

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The Key-Value Storage Model

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Granting Access to the Key-Value Store

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The WIT Contract Behind the SDK

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The Rust Key-Value API

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Why Key-Value Is the Default Choice

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The NoOps Model: Spin, Akamai Functions, and Kubernetes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Browsing the Key-Value Store Using kv-explorer

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Key-Value vs. Relational Storage

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Task-Engine: The End-to-End Application

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Building the Task-Engine

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Scaffolding Multi-Component Applications

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Creating a GitHub Repository

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Task-Engine Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Component Blueprint

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The **task-api**: Core HTTP API

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The **frontend**: Static File Server

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The **classifier**: Internal Domain Logic

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The **worker**: Scheduled Maintenance

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Architectural Rationale: Component Separation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Implementing the `task-api` Component

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Adding the `task-api` Component

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Creating Rust Modules

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Isolating the Data Models

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Decoupling the Key-Value Store

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Implementing the Business Logic

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The Entry Point

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Granting Access to the Key-Value Store

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Compiling and Testing the API

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Building and Serving the Frontend

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The Static File Server Component

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Downloading the Frontend Assets

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The Frontend Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Static File Server Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Testing the Frontend

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Integrating External APIs: Adding Notifications

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Granting Outbound HTTP Capabilities

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Defining the Spin Variables

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Loading Configuration into Router State

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The Notification Model

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Defining the Notifier Trait

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Implementing the NtfyNotifier

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Extracting State in Route Handlers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Sending the Notifications

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Verifying the Notification Flow

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Component Composition: The `classifier`

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Component Interaction Models

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Scaffolding the `classifier` Component

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Defining Component Dependencies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Generating Bindings in `task-api`

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Updating the API and Domain Models

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Invoking the Classifier from `task-api`

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Build and Verify

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Implementing the Background Worker

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Choosing HTTP for Worker Portability

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Adding the Worker Component

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Gating Endpoint Access with a Shared Secret

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Validating Worker Requests

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Cleanup and Report Generation Logic

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Testing the Background Worker

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Operational Excellence

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Structured Logging

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Where Spin Sends Component Output

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Why Plain Text Stops Scaling

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Adding Structured Logging to `task-api`

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Controlling Log Levels and Output Format

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Continuous Integration with GitHub Actions

Improving the Task-Engine means we should have some sort of gatekeeper. A system that verifies that recent code changes are syntactically correct by recompiling the entire application in a clean environment (a machine other than your local development environment), and that means Continuous Integration (or CI for short).

For a Spin app, the CI job is always the same shape: install the language toolchains your components need, install Spin, and run `spin build`. Writing that YAML by hand is tedious and easy to get subtly wrong: a forgotten Wasm target, a pinned Spin version that drifts from your machine, or a toolchain step for a language you don't even use.

Although I faked the commits for the following screenshot, I bet you've been there, or at least stumbled upon a list of commits like this:

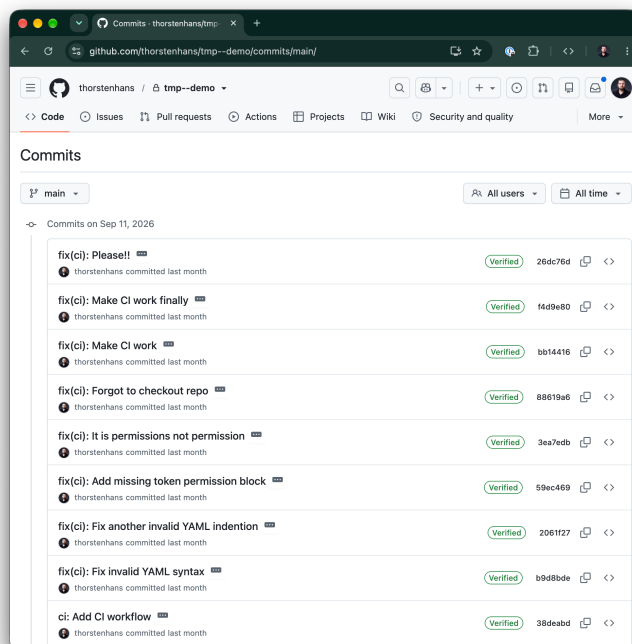


Figure 6.1. A pretty common Git log

I don't want to send you down a similar route. Instead, I want to give you the tool that makes generating these rather complex GitHub Actions workflow files a breeze. The `gh` plugin for Spin is exactly that tool. It can inspect the components of Spin applications and generate a tailored, ready-to-commit GitHub Actions workflow for you (optionally with continuous-deployment steps, which we'll address in the following chapters as well).

Installing the Plugin

Plugins come from Spin's plugin catalog. Refresh your local copy of the catalog, then install `gh`:

```
# Update Spin plugin catalog
spin plugins update

# Install the gh plugin for Spin
spin plugins install gh --yes
```

That's it – `spin gh` is now a first-class subcommand.

How `spin gh` Works

Point `spin gh` at a folder, and it recursively scans that folder and every subfolder, detecting all the Spin apps it finds and every component inside them, across Rust, Go, JavaScript, TypeScript, and Python. From that inventory, it generates a GitHub Actions workflow that:

- Checks out the actual code
- Installs the required language toolchain for each component it finds (and only those)
- Installs Spin
- Runs `spin build` for each Spin application discovered

Two subcommands do the work:

- `create-action` - Generates the GitHub Actions workflow file.
- `eject` - Renders the plugin's default template to disk so you can customize it.

Generating a Workflow

For generating a new GitHub Action workflow file, we will use the `create-action` command. It has a lot of flags, but they group around three core concepts:

GitHub Action customization:

Flag	Description	Default
ci <branch>	Run on push to a specific branch	main
pr <branch>	Run on PRs targeting a branch	
cron "<expr>"	Run on a cron schedule	
manual	Add a workflow_dispatch trigger	
dry-run	Print to stdout instead of writing a file	
overwrite	Overwrite if output file exists	.github/workflows/ci.yaml
output	Workflow file output path	
runner-os	Runner OS	
env	Custom environment variables	
template	Custom template	

Tool & toolchain version management:

Flag	Description	Default
spin-version	Pin the Spin Version	latest
spin-plugin	Spin plugins that should be installed	
rust-version	Pin the Rust Version	1.98.1
rust-target	Specify Rust compilation targets	wasm32-wasip1
go-version	Pin the Go Version	1.26.6
node-version	Pin the Node.js Version	26
python-version	Pin the Python Version	3.13.0

Deployment:

Flag	Description
deploy-to-akamai-functions	Deploy to Akamai Functions
push-oci-artifacts	Build & publish OCI artifacts
sbom	Generate SBOM for OCI artifacts
sign	Sign OCI artifacts

Let's generate a workflow for Task-Engine, which should run whenever we push to main. By appending the `--dry-run` flag, we can review what would be generated instead of the file actually being written to disk:

```
# Generate the GitHub Action
spin gh create-action \
  --name "CI Task-Engine" \
  --ci main \
  --dry-run
```

This command will come back in a matter of seconds and show how the full-fledged CI pipeline could look like:

```
name: "CI Task-Engine"
on:
  push:
    branches:
      - "main"
env:
  RUST_VERSION: "1.98.1"
  RUST_TARGET: "wasm32-wasip2"
jobs:
  spin:
    runs-on: "ubuntu-latest"
    name: Build Spin App
    permissions:
      contents: read
    steps:
      - uses: actions/checkout@v4
```

```
- name: Install Rust
  uses: dtolnay/rust-toolchain@stable
  with:
    toolchain: "${{ env.RUST_VERSION }}"
    targets: "${{ env.RUST_TARGET }}"
- name: Install Spin
  uses: fermyon/actions/spin/setup@v1
  with:
    github_token: ${ secrets.GITHUB_TOKEN }
- name: Build task-engine
  run: spin build
  working-directory: .
```

Notice what the plugin worked out on its own: it detected at least one Rust component and added exactly one toolchain step (dtolnay/rust-toolchain with the Wasm target), then an Install Spin step, then spin build.

No installation of Go, Node, or Python, because there's nothing here that would require these tools. Run the command again, but drop the `--dry-run` flag, and it will write the exact same content to `.github/workflows/ci.yaml`.

Commit that file, and push to your main branch. A few seconds later, you should see your first successful CI on GitHub:

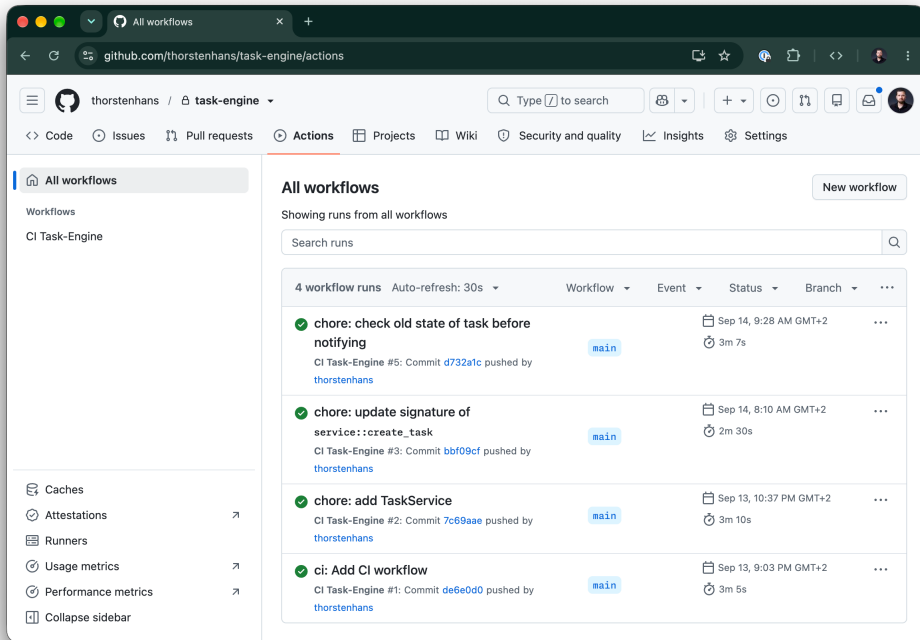


Figure 6.2. GitHub Action generated by `spin gh`

That's Continuous Integration handled with a single command. Next, we dive into testing!

Unit Testing Business Logic

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Why Test Outside the Wasm Boundary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Let the Router Own the Routing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Mocking the Classifier

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Mocking the Notifier

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Mocking the Repository

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Testing the TaskService

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Running the Tests

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Integration Testing Spin Apps

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The Shape of the Harness

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Implementing SpinTestBed

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Integration Testing task-api

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Integration Testing the worker

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Running the Tests

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Going Live

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Securing and Hardening Your Application

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Hardening Spin Manifests

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Establishing the Baseline: Capability Grant Assessment

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Five Manifest Hardening Techniques

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

1. Outbound Networking: Don't Use Wildcards

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

2. Restrict Filesystem Mount Points

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

3. Restrict Variable Scoping and Component Mapping

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

4. Narrow Your Triggers and Routes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

5. Use Private Endpoints for Internal Services

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Component Dependency Capabilities

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The Software Supply-Chain Dilemma

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Zero-Trust Composition by Default

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Controlling Capability Inheritance

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

1. Deny All (Default)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

2. Selective Capability Inheritance

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

3. Full Inheritance

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

4. Blanket Dependency Inheritance

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Hardening Application Code

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Adding Request Payload Validation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Enforcing Payload Size Limits and API Security Headers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Signing and Verifying Spin Applications with cosign

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Signing and Verifying on Your Local Machine

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Generating a Local Key Pair

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Pushing Task-Engine to a Registry

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Signing the Task-Engine OCI Artifact

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Verifying the Artifact

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Automating Keyless Signing in GitHub Actions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Verifying Keyless Artifacts Locally

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Gating `spin` up on Signature Verification

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Software Bill of Materials (SBOM)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The WebAssembly Advantage: Eliminating OS Noise

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Rust Lockfiles and License Scanning Limitations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Local SBOM Generation with Trivy

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Local SBOM Scanning with Trivy

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Adding an Automated CI Quality Gate

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Distributing the SBOM with ORAS

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Creating, Scanning, and Publishing SBOMs with GitHub Actions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

To the Global Edge with Akamai Functions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

What Is Akamai Functions?

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Architectural Benefits of the Edge

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

When the Edge Is a Strong Fit

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

When to Reconsider

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Constraints: The HTTP-Only Nature of the Edge

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Task-Engine Deployment to Akamai Functions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Installing the aka Plugin

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Authorizing the Spin CLI

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Building and Deploying

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

What Happens When You Deploy

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Recurring Deployments and Workspace Linking

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Navigating Your Akamai Functions Account

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Variable Persistence on Akamai Functions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Day-Two Operations with Akamai Functions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Data at the Edge: State in Distributed Environments

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Akamai Functions' Managed Key-Value Store

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Quotas and Limits

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Scheduling the Background Worker

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Creating a Cron Job

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Managing Cron Jobs

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Continuous Deployment with GitHub Actions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Generating the Deployment Workflow

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Configuring Deployment Variables

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Creating a Personal Access Token

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Configuring GitHub Repository Secrets

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Verifying Continuous Deployment

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Running WebAssembly on Kubernetes with SpinKube

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The SpinKube Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The Node Level: Bypassing the Container

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The Control Plane: The Spin Operator

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Separation of Concerns: `SpinApp` and `SpinAppExecutor`

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Cluster Setup and SpinKube Installation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Why k3d?

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Installing SpinKube

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Installing Application Dependencies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Creating the SpinAppExecutor

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Deploying Valkey

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Creating Application Secrets

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The SpinApp and SpinAppExecutor CRDs

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The Minimal SpinApp Spec

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Configuring Variables and Runtime Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The SpinAppExecutor CRD

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Deploying the Task-Engine SpinApp

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Creating the SpinAppExecutor Manifest

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Locating the Task-Engine OCI Reference

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Creating the Image Pull Secret

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Scaffolding the SpinApp Manifest

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Configuring Secrets and Valkey Integration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Applying and Verifying the Deployment

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Port-Forwarding and Testing the Application

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Scheduling the Background Worker

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Continuous Deployment with GitHub Actions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Configuring Secrets and Cluster Access

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Extending the GitHub Actions Workflow

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Make Spin Your Own

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Extending Spin

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Authoring and Distributing Custom Spin Templates

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Anatomy of a Spin Template

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Authoring the Template Manifest

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Supporting `spin add` with Component Snippets

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Authoring Content Files

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Scaffolding Dependencies with `Cargo.toml.tpl`

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Embedding Hardened Service Boilerplate

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Handling Standalone Applications with `content/spin.toml`

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Installing and Using the Template

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Distributing Templates

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Authoring Functional Spin Plugins

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The Plugin Execution Model

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Building `spin bump` in Rust

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Streamlining Local Installation with `pluginify`

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Testing the Plugin with Task-Engine

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Packaging and Distribution

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Authoring Custom Trigger Plugins

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The Trigger Plugin Execution Model

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The Challenge: Managing Startup Dependencies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

The Trigger Plugin Discovery Contract

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Implementing the Trigger in Rust

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Packaging and Installing the Trigger Locally

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Running an Application with the Custom Trigger

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Contributing Upstream

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Contributing Beyond Runtime Code

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Navigating the Spin Ecosystem

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Connecting on CNCF Slack

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Participating in Community Meetings

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Making Your First Contribution

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Real-World Spin Apps

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Practical Architectural Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Pre-Initialization with Wizer

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Refreshing Data with GitHub Actions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Sandboxed Agent Tools with MCP

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

LLM-Powered Chatbots

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Declaring Required Capabilities

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Coordinating the Chat Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Dynamic Image Processing at the Edge

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Resizing and Transcoding in Memory

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Anti-Scraping Through Edge HTML Rewriting

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

In-Flight DOM Transformation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Intercepting and Rewriting Upstream HTML

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Edge API Aggregation and Fan-Out

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Authorizing Upstream Hosts

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Concurrent Fan-Out and Merging

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.

Bringing It All Together

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/building-serverless-apps-with-wasm-and-spin>.