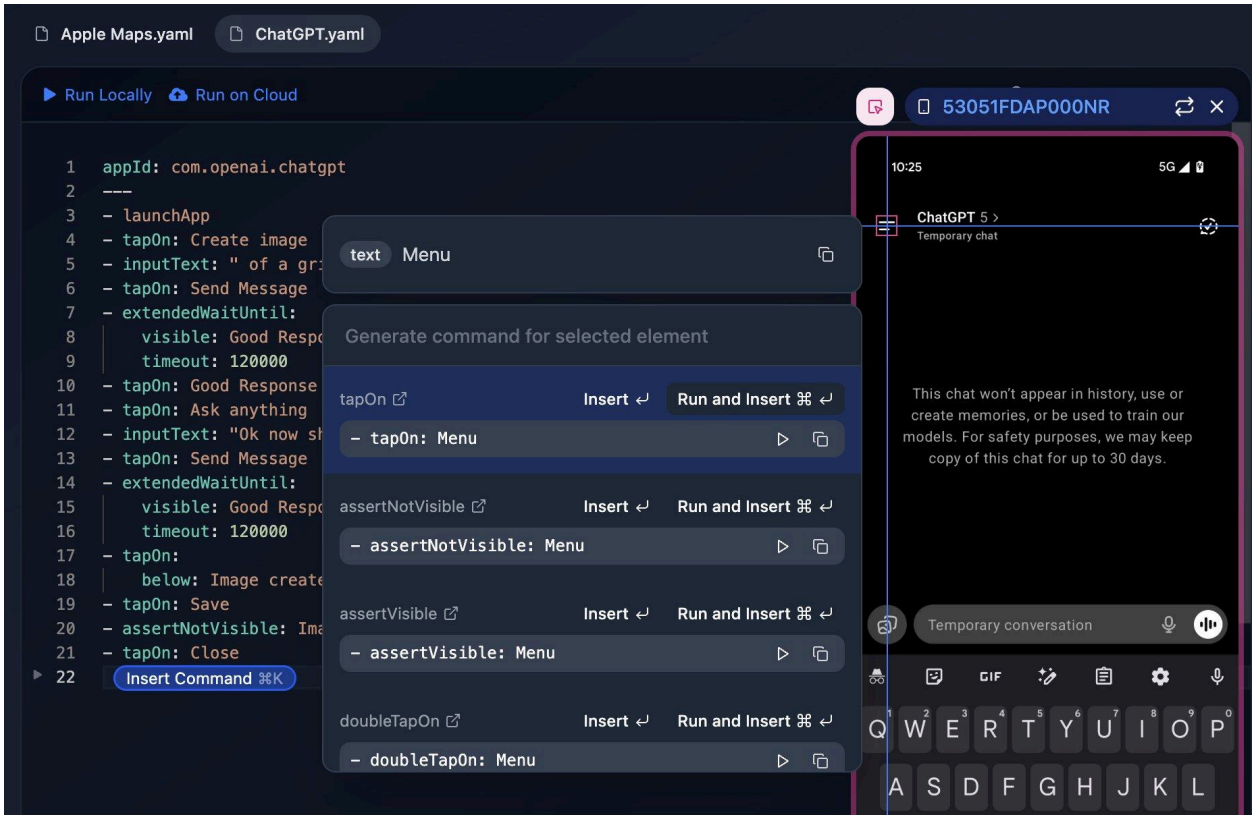
How Maestro is Reinventing Mobile Test Automation

Leland Takamine



At Maestro, we're completely reinventing mobile test automation - and eventually, automation in general.

It's a bold claim, but we have some of the deepest mobile expertise in the industry. And with that, we've built a completely novel architecture that powers the simplest, most loved, and most universal open-source test automation framework. Today, Maestro is being adopted by companies like Microsoft, Meta, and DoorDash - who all now prefer Maestro over long-standing incumbents like Appium, Detox, and Playwright.

We're still just getting started, but it's a good time to reflect on the principles, decisions, and technical foundations that got us here.

Here's how we did it 👇

No Setup, No Dependencies, No Driver Installations, No SDK

This is a principle we committed to from day one.

Maestro is a complete testing framework that can be installed in one line with no dependencies, no driver installation steps, and requires no SDK.

In order to achieve this, we needed to build from the ground up.

As part of this effort, we contributed dadb to the Android ecosystem. Dadb allows programmatic communication with Android devices without requiring a separate ADB server process. Other testing tools build on top of a standalone ADB binary, requiring end users to install the adb CLI separately. By implementing dadb - a pure Kotlin implementation of the ADB protocol - we completely eliminated the reliance on this external process (ADB Server), improving both the user experience and the reliability of the system. Dadb is now a fundamental building block that powers the Maestro framework and Maestro Cloud.

We've also gone deep into the internals of the mobile operating systems to provide robust, self-installing drivers with zero dependencies. See Maestro - Re-Building the iOS Driver.

The result? A single line install command.

Compare that to Appium, where the first step is reading a dissertation on all the components involved, then installing 3 separate things that all require specific versions to work together. 🤮

With Maestro, you can go from never hearing about Maestro to running your first test in 5 minutes.

Universal Compatibility Across All Platforms and Frameworks

Native, React Native, Flutter, Ionic, or whatever app framework you're using - they're all supported by Maestro. That's by design.

Maestro interacts primarily with the accessibility layer of the device - the same one used by screen readers like VoiceOver and TalkBack. This architectural decision ensures Maestro can automate ANY application, debug and production builds alike, regardless of the app framework.

And with the mobile ecosystem leaning more and more into Maestro, compatibility will only continue to deepen and improve. Frameworks like React Native and Flutter are making changes specifically to improve the Maestro testing experience - in some cases even using Maestro to test the frameworks themselves.

Here are few key ecosystem enhancements driven by Maestro:

We were the ones who added resource ids to Flutter (the most upvoted testing issue in the Flutter core repo).

- Making Maestro work better with Flutter

A Maestro community member drastically optimized React Native's accessibility tree to speed up Maestro iOS execution by 4x.

- PR: Only generate recursive accessibility label for accessible elements

The most popular React Native framework, Expo, has embraced Maestro as the preferred testing platform, providing first-class support within their platform.

- Run E2E tests on EAS Workflows and Maestro - Expo Documentation

And perhaps the strongest signal that the ecosystem is fully embracing Maestro:

Meta has fully adopted Maestro for end-to-end testing of the React Native framework itself!

- React Native's Maestro test suite

Human-Readable Syntax (That Isn't a Gimmick)

Teams often view end-to-end testing as an engineering challenge - something to engineer their way out of. "Maybe a more sophisticated test harness will solve our problems?"

In reality, end-to-end testing serves one purpose: to ensure your product functions as intended. This is a business need, and it requires a no-BS solution that just works. As soon as you lose sight of this, you'll end up with an unmaintainable, flakey test suite. Here's the litmus test: Does your test suite feel like a software engineering project in itself? Then you're either doing it wrong - or more likely, you're using the wrong tools.

The choice of YAML for Maestro's language was met with skepticism from some folks, which is reasonable if you've been burned by complex infrastructure-as-code solutions. But with Maestro, your flows are a simple flat list of commands - YAML is actually great for this:

```
None

appId: com.example.app
---
- launchApp
- tapOn: "Login"
- inputText: "user@example.com"
- tapOn: "Password"
- inputText: "password123"
- tapOn: "Sign In"
- assertVisible: "Welcome"
```

More importantly, Maestro's syntax prevents you from being lured into the trap of over-engineering. It's anti-spaghetti code by default, yet powerful enough that the most sophisticated teams in the world like Microsoft Copilot, DoorDash, Meta are all using Maestro to test their applications.

Tooling That Levels Up, Instead of Dumbing Down

Here's something we believe companies are getting wrong today: They confuse the goal of making people more efficient with the assumption that users don't know what they're doing. At Maestro, we strive to respect the intelligence of our users and build tooling that truly empowers them to do their jobs.

At the core of the Maestro test framework is a concept we call the Maestro "hierarchy". It’s the raw representation of the screen that Maestro sees and interacts with. It's messy, but it's also the most foolproof way to understand what's happening when writing or debugging a script. So instead of hiding this complexity from users, our tooling (eg. Maestro Studio and `maestro hierarchy`) makes the hierarchy MORE visible, allowing our users to dive deep when they need to.

The combination of Maestro's human-readable syntax and powerful, yet accessible tooling is what has finally opened up *reliable* automation testing to testers with little to no technical background. This is a game changer for our users, who often tell us stories about how they've upskilled their manual testers to write automated tests using Maestro, freeing up their developers' time and enabling more frequent releases. In one memorable case, this took only a single week of training and allowed the whole company to move from monthly releases to weekly.

And Yes... AI

Let's address the elephant in the room. AI testing is the hot new buzzword, with natural language testing startups being founded every week. And like all hype cycles, most of it is nonsense. As one Reddit user put it:

Every time I try one of these "low code"/"no code" automation/AI software programs, I do not feel like I am getting the job done quickly. I feel like it's taking me more time to set up simple tests than it would be if I were to just write them myself in a preferred language & framework... I notice I am waiting a lot just for a couple of steps to run just to verify that it's even working accurately.

But we're still 1000% bullish on AI - when applied strategically, and for features that play to its strengths.

And to be transparent, we weren't immune to the hype either initially. We took our stab at full natural language AI testing, spending 6 months building what was the leading natural language testing tool at the time - but ultimately shut down that product. Why? AI natural language testing makes for a Super Cool Demo™, but isn't actually what's most useful for real users. We tried

AI-generated code under the hood too. The result? A slower, more frustrating workflow than simply writing Maestro code yourself.

So what does work?

Our most recent AI feature: Maestro MCP. It makes everything Maestro is great at available to Claude, Codex, Cursor, or whatever AI tool you prefer. But still outputs Maestro code that is fully readable and deterministic. This is a good representation of our approach to AI going forward: Leverage AI to make humans more efficient, while still respecting the user and providing full control of the output. And that same declarative syntax that our users love? It's great for AI too. Maestro raises the level of abstraction, providing the automation building block that superpowers both humans and AI.

We Believe in Strong Foundations

Here's what sets Maestro apart: We've spent years building a better foundation for automation. Owning the full stack - from the framework to the IDE to the cloud platform - allows us to be uncompromising when it comes to reliability and user experience. This is why so many teams have come to love Maestro, and this is exactly what we plan on doubling down on going forward.

The future of test automation isn't about flashy demos or buzzword-driven solutions. It's about building robust tooling that respects developers' and testers' intelligence, while dramatically improving their productivity. That's what we're building at Maestro - and we're just getting started.