How we're making installation issues a thing of the past

Update: Since this blog post was published, we have releasedGanache a personal blockchain and a replacement to the TestRPC. We have left this post unaltered, but we highly recommend checking out our Ganache Quickstart page.

It doesn't matter what operating system you're on -- if you've used Truffle in any capacity, you've likely run into an install issue in some form or another. This is especially true for Windows, as Windows doesn't generally ship with the necessary tools to install all of Truffle's important dependencies. And for most users, this sucks: getting hung up on installation is incredibly frustrating when you're usually just one error message or two away from creating your first Ethereum-enabled application. Well, I've got good news. As of the latest Truffle beta release, I'm happy to tell you that this is an issue no longer.

That's right: No matter what platform you're on, your install issues should be over. This is big.

In order to completely sidestep most if not all of the most common installation issues, we had to overcome some technical difficulties as well as rethink how we ship our code. In this post, I'm going to tell you about what we did and how that made installation incredibly simpler, on every platform, for everyone.

So what was the problem?¶

Since Truffle is a Javascript-based application, we (naturally) ship our code using theNode Package Manager (NPM for short). NPM hosts thousands if not millions of code packages, and it's the service that makes modern Javascript programming possible. Without NPM, not only would programming Truffle be nearly impossible, but programminganything complex in Javascript would be impossible: NPM lets developers separate and bundle code into its own individual chunks, called "packages", that any of those other chunks (packages) can then rely on.

Because of NPM, Truffle is standing on the shoulders of giants. Truffle relies on the code of so many other people that if you looked at the full list you'd be hard pressed to give us credit for the codebase. And that's the beauty of it: packages like hashing functions and cryptographic libraries -- hard libraries to write (and get right!) -- we don't have to write ourselves.

Though using dependencies on NPM has significant upsides, it also comes with significant drawbacks. Those same hashing functions -- functions used throughout Ethereum -- are written to be extremely performant, and so are written in C. In order to install those packages, NPM has to be able to compile them on the fly during installation, which means every Truffle user needs to have a C compiler installed. Additionally, NPM (for some reason) uses Python during the compilation process, which means Truffle usersalso need to have Python installed.

Bottom line: In order to install a Javascript project, Truffle users need both C and PythonThis is madness.

So what can we do?¶

Most of the issues people were seeing stem from the compilation process mentioned: Either they didn't have the right compilers/interpreters installed for C and Python, or C and Pythonitself required some dependency that wasn't properly installed. We've had atutorial written for awhile that shows Windows users how to correctly set up their environment and get everything installed, but this required considerable effort from novice users and didn't take into account the many landmines hiding around every corner, waiting to explode at any moment. So we can't "instruct" our way around this problem. The only way to fix it, then, is to sidestep it completely.

Sidestepping the compilation process means completely excising any compilation out of the installation process. The way we chose to tackle this is two ways:

- Remove any unnecessary dependencies that rely on compilation.
- Replace dependencies that need compilation with versions 100% written in Javascript.

In theory this sounds easy: i.e., "just use the Javascript versions or don't use the C versions as all". In practice, this is much harder. Often, we don't directly control the dependencies our application relies on, becausedependencies of dependencies (of dependencies of dependencies) might rely on a package that requires compilation. Since the bad apples are the packages that power the code we rely on (i.e., they're "indirect dependencies"), we have to surgically remove them and replace them with versions that install and function more easily. This is no small matter.

Thankfully, a tool exists that makes this surgical replacement possible. It's calledWebpack . Webpack bills itself as a Javascript bundler, aiming to mash together all your project's dependencies into a single blob of code, i.e., a "bundle". Webpack finds most of its uses in browser applications, as browsers need to download all code first before a website can be displayed, and it's more performant for the browser to download that code in a single request. However, for console applications like Truffle, Webpack can still be immensely valuable. By creating a bundle with Webpack, we can control exactly which dependencies get included in it.

Takejs-scrypt for instance. This library provides hashing functions required by Ethereum, is 100% written in Javascript, and is API-compatible with the C-compilednode-scrypt library. With Webpack, we can simply removescrypt and replace it withjs-

scrypt like nothing happened.

How about another example: Let's do IPFS. Truffle relies on theethpm-js package to provide EthPM support, which itself relies on theipfs package to communicate with IPFS servers. Somewhere deep down inipfs 's dependency tree, some code needs theexecSync package, a package that causes installation issues for some users. Using Webpack, when creating our bundle, we can justnot include it . Since that code only matters for parts of IPFS that Truffle doesn't use, that code doesn't need to be in our bundle. Ex-nay,gone-yay . Zip, zilch, completely gone.

Creating a bundle for our releases instead of letting NPM manage dependencies has other benefits as well. To keep code modular and maintainable, Truffle relies on almost 2000 packages! (This includes the dependencies of dependencies we mentioned before). Bundling ahead of time creates a single file users have to download during installation instead of requiring NPM to make requests for all 2000 dependencies. So not only do we get to sidestep all of the installation issues, but we make installation nearly instant as well.

Yep. Feel free to party. We did.

So how can you get it?¶

Bundling Truffle's dependencies means releasing a version of the code where certain dependencies have been either replaced or removed. This not only means releasing structurally new code, but it means releasing code that has differing testability characteristics which change our ability to ensure its of the quality you've come to expect. We need to build infrastructure to help us do that, so in the meantimewe've released the new version of the code in beta.

To get the beta version of Truffle, first uninstall the version you currently have (they'll conflict):

npm uninstall -g truffle And now install the beta version:

npm install -g truffle@beta You'll notice that the beta version installs almost instantly, and no compilation is required. Hot-dog.

So what's next?¶

As we gain responses from the community about this new version we'll likely make changes to ensure we've sidestepped all the installation issues that have plagued users in the past. Once that's finished we'll make an official release. As well, we'll be performing this same work for the TestRPC in the coming weeks to ensure the TestRPC has the same hassle-free, instant installation that you'll soon come to expect from Truffle.

If you run into issues with the beta version, don't hesitate towrite up an issue in our issue tracker or reach out on thecommunity GitHub Discussions channel , where hundreds of your fellow Trufflers congregate to answer your questions.

Cheers, and we appreciate your ongoing support and feedback.

-- Tim & the Truffle Team