Goodbye, Dependency Installations

Author: Louis Cruz

Claps: 143

Date: Feb 7

At Klaviyo, our frontend codebase is a large monorepo with dozens of teams actively contributing to it. This monorepo comprises more than 260 packages, almost entirely in TypeScript. Every commit pushed undergoes a wide swath of validations in continuous integration, and every change to the primary branch is continuously deployed. Hundreds of commits are pushed a day across all branches, and itâ€™s not uncommon for three dozen releases to make their way to production on a given workday.

Development velocity is important to us. Given the size and activity level of the repository, and given that the monorepo has more than 3,800 external dependencies, it is also a constant challenge. Continuous integration times determine the speed at which we can mitigate issues in production, the rate at which we can release new features, and the quality of life of engineers making contributions. On top of that, Node has quirks in its module resolution strategy. These can cause significant hurdles in reasoning about how a given dependency will be resolved at runtime in a complex monorepo like ours, making it difficult to make changes safely.

Early in 2022, we decided to put effort into improving developer experience. An obvious target at the time was to reduce our Docker build times during continuous integration runs. In cases where most of the Docker image build layers had a cache hit, it might only take a few minutes for the Docker build to complete before the important CI stuff could begin. But in the case that most or all of the layers had a cache miss, the Docker build would take eleven minutes in the average case and sixteen minutes in the worst case.

The vast majority of this time was spent on dependency installation, which accounted for roughly half of the total CI time. This scenario was not infrequent and often happened at the worst times. Any change that modified external dependencies or inter-monorepo dependencies in any way would encounter this scenario.

Since our monorepo uses Yarn as its dependency manager, we thought it would be a good opportunity to upgrade to a modern Yarn version and then try migrating to Yarn Plugâ€™nâ€™Play, often referred to as Yarn PnP, since it could prevent the need for installing dependencies at all.

On a personal note: Iâ€™ve heard quite a few engineers mention that they would like to make use of Yarn PnP to remove the need for dependency installations, but they assume it would be too much work to actually accomplish. Since we have successfully accomplished this at Klaviyo for our large frontend codebase, I wanted to provide a case study for other engineering organizations to consider.