Migrating to Meteor 1.8

Most of the new features in Meteor 1.8 are either applied directly behind the scenes (in a backwards compatible manner) or are opt-in. For a complete breakdown of the changes, please refer to the changelog.

The above being said, there is one required migration step and few things that you should note.

Update the `@babel/runtime`

Update the `@babel/runtime` npm package to version 7.0.0 or later:

```
meteor npm install @babel/runtime@latest
```

web.browser.legacy

Meteor 1.7 introduced a new client bundle called `web.browser.legacy` in addition to the `web.browser` (modern) and `web.cordova` bundles. Naturally, this extra bundle increased client (re)build times. Since developers spend most of their time testing the modern bundle in development, and the legacy bundle mostly provides a safe fallback in production, Meteor 1.8 cleverly postpones building the legacy bundle until just after the development server restarts, so that development can continue as soon as the modern bundle has finished building. Since the legacy build happens during a time when the build process would otherwise be completely idle, the impact of the legacy build on server performance is minimal. Nevertheless, the legacy bundle still gets rebuilt regularly, so any legacy build errors will be surfaced in a timely fashion, and legacy clients can test the new legacy bundle by waiting a bit longer than modern clients. Applications using the `autoupdate` or `hot-code-push` packages will reload modern and legacy clients independently, once each new bundle becomes available.

Overriding package version

The `.meteor/packages` file supports a new syntax for overriding problematic version constraints from packages you do not control.

If a package version constraint in `.meteor/packages` ends with a `!` character, any other (non-`!`) constraints on that package elsewhere in the application will be *weakened* to allow any version greater than or equal to the constraint, even if the major/minor versions do not match.

For example, using both CoffeeScript 2 and `practicalmeteor:mocha` used to be impossible (or at least very difficult) because of this `api.versionsFrom("1.3")` statement, which unfortunately constrained the `coffeescript` package to version 1.x. In Meteor 1.8, if you want to update `coffeescript` to 2.x, you can relax the `practicalmeteor:mocha` constraint by putting `coffeescript@2.2.1_1! # note the !` in your `.meteor/packages` file. The `coffeescript` version still needs to be at least 1.x, so that `practicalmeteor:mocha` can count on that minimum. However, `practicalmeteor:mocha` will no longer constrain the major version of `coffeescript`, so `coffeescript@2.2.1_1` will work.

Migrating from a version older than 1.7?

If you're migrating from a version of Meteor older than Meteor 1.7, there may be important considerations not listed in this guide (which specifically covers 1.7 to 1.8). Please review the older migration guides for details:

- Migrating to Meteor 1.7 (from 1.6)
- Migrating to Meteor 1.6 (from 1.5)
- Migrating to Meteor 1.5 (from 1.4)
- Migrating to Meteor 1.4 (from 1.3)
- Migrating to Meteor 1.3 (from 1.2)