Migrating to Meteor 1.3

Breaking changes

These are all the *breaking changes* – that is changes that you absolutely have to worry about if you are updating your app from 1.2.x to 1.3. However, we recommend that you also consider the *recommended* changes listed in the other sections below.

- Ensure that your project has a `package.json` file, which will be the foundation for npm package installs. You can create this by running: `meteor npm init -y`
- Files in a directory named `imports/` will no longer load eagerly. (You should probably rename such a directory as it is the basis of our new module system).
- Files within your app named `*.test[s].*`, `*.app-test[s].*`, `*.spec[s].*` and `*.app-spec[s].*` will no longer load eagerly (you should probably rename such a file if it doesn't contain tests, as it will be eagerly loaded by our new app testing modes).
- If you are using React you will now need to install a set of React npm packages in your app. See the recommendations for React below for more details.

Mobile

- iOS apps now require iOS 8 or higher, and building for iOS requires Xcode 7.2 or higher to be installed.
- Building for Android now requires Android SDK 23 to be installed. You may also need to create a new AVD for the emulator.
- Cordova has been upgraded to the most recent versions (Cordova 6.0.0, Cordova iOS 4.0.1 and Cordova Android 5.1.0). This may require you to upgrade your plugin versions. We pin core Cordova plugins to versions known to be compatible and warn about this during build, but you may encounter compile time or runtime errors with third party plugins. Upgrading to newer versions of these plugins may help if they have been updated to work with recent versions of Cordova.

- The plugin used to serve your app's files and support hot code push has been completely rewritten. As a result, files are now served from `localhost` instead of `meteor.local`, with a fixed port number derived from your `appId`. You may have to update OAuth redirect URLs to point to the new local domain and port.

Recommendations: modules

The biggest new feature in Meteor 1.3 is support for ES2015 modules on the client and the server. Using modules you can declare dependencies between files, control load order, and use npm packages on the client and server.

- You should load all Meteor "pseudo-globals" using the `import { Name } from 'meteor/package'` syntax. For instance:

```
import { Meteor } from 'meteor/meteor';
import { EJSON } from 'meteor/ejson';
```

- You should consider installing the `meteor-node-stubs` npm package to allow using npm packages written for `node` on the browser:

```
meteor npm install --save meteor-node-stubs
```

- If you are using app-local packages to control load order and write unit tests for your application, we recommend you switch to using modules:
- Remove code related to the Package API from the `package.js` files and rename them to `index.js`,
- Move your local packages to the `imports/` directory.
- Add the necessary `import` statements to each of the modules in your packages.
- Add `export` statements to each of your packages exports.

```
api.addFiles('file.js');
// For files that are not imported elsewhere, this turns into
import './file.js';

// Remove from package.js
api.export('Foo');

// localPackage/foo.js
// Foo must be explicitly exported
export default Foo;

// client/main.js
import '/imports/localPackage';
```

- You can read about our recommended structure for applications and modules in the Application Structure article of the Meteor Guide, and how to test them in the Testing article.

- If you are using Atmosphere packages which wrap npm packages, both on the client and server, it is now recommended that you install them using npm. Run `npm init` to initialize your `package.json` and install packages with `npm install --save` (or `npm install --save-dev` if it's a development dependency for testing etc.). We have some tips about how to use npm packages written in an asynchronous style.

Also, you should no longer need to use the `meteorhacks:npm` package. To migrate, follow the following steps:

1. Remove packages from your app: `meteor remove meteorhacks:npm npm-container`.
2. Remove the generated `npm-container` package: `rm -r packages/npm-container`.
3. Move the contents of `packages.json` to the `dependencies` section of your `package.json` (you may need to create one with `meteor npm init`).
4. Use `import` instead of `Npm.require()`.

Recommendations: package authors

Package authors are recommended to:

- No longer publish wrapper packages that do no more than include an npm package / client side lib. If your package adds significant wrappers around the npm package, it might make sense however.
- Publish to npm when appropriate, especially if your package can be used by the wider JS community!
- Use `api.mainModule()` and `export` from your main module rather than `api.exports()` in Atmosphere packages.
- If you depend (directly or transitively) on a client side npm package that is large or problematic if installed twice (e.g. React), use `tmeasday:check-npm-versions` to declare "peer" dependencies. If the client side npm package you depend on is `angular`, you can support both Meteor 1.2 and 1.3 using this solution. Read more about this in the Writing Packages article.

Recommendations: Testing

Meteor 1.3 includes a new command `meteor test`, which can be used to run tests against your app, in two modalities. You can read about these features in much more detail in the Testing Guide Article.

Full app testing

If you were previously using Velocity to run tests against your running Meteor app, the full app test mode should allow you to run your tests against 1.3, with some small changes.

- To convert tests, you'll need to change or upgrade your test driver package to a 1.3 compatible package (as of this writing there is only one choice

`practicalmeteor:mocha` but we expect more to exist in the future). You should name your test files in the pattern `*.app-test[s].*` and place them *outside* of `tests/` directories. To run the tests you can run `meteor test --full-app --driver-package <driver-package>`

- Note that full app test mode does not run the test reporter in a separate application to the app under test, and does not amalgamate results from multiple testing systems, as Velocity does. This effectively means if you are using more than one testing system, you will need to run `meteor test --full-app` multiple times.
- Also, it means certain types of tests are better off written as *acceptance tests* outside of the Meteor tool.

Module testing

If you were previously using in-app packages in order to unit test your app, you should switch to a modules-based approach and test them using the normal test mode.

- To convert your unit tests to run against the app, first upgrade your test driver (see above) and then place your test files alongside the modules they are testing with a name matching `*.tests.*`. Such files will automatically be added to your "test app" when you run `meteor test --driver-package <driver-package>`. You can `import` the modules that you need to test against within each test file.
- Some example tests can be seen the Todos example app

Recommendations: Mobile

Alongside some of the breaking mobile changes listed above, there are some changes in the way the mobile integration works that you should consider:

- Some low resolution app icon and launch images sizes for now unsupported devices have been deprecated. To avoid a deprecation warning during build, please remove the entries from your `mobile-config.js`. (You probably also want to remove the image files from your project.)
- The plugin now allows for local file access on both iOS and Android. You can construct file system URLs manually (`http://localhost:<port>/local-filesystem/<path>`) or use `WebAppLocalServer.localFileSystemUrl()` to convert a `file://` URL.

Install React from npm

In Meteor 1.3, we recommend installing `react` and `react-dom` into your app using npm, and importing them from your app code:

```
import React from 'react';
import ReactDOM from 'react-dom';
```

As mentioned in the breaking changes, the `react` Atmosphere package still works, but it now expects you to install the React npm packages it uses in your application (read the Using Packages article for more details about how to manage your npm dependencies):

```
npm install --save react react-dom react-addons-transition-group \
 react-addons-css-transition-group react-addons-linked-state-mixin \
 react-addons-create-fragment react-addons-update react-addons-pure-render-mixin \
 react-addons-test-utils react-addons-perf
```

However, we recommend that you should stop using the `react` or `react-runtime` Atmosphere packages and instead install React directly from npm (for more detail, see the React article of the guide). To make this change in an existing app, you can run:

```
meteor remove react

# if you are using our data integration
meteor add react-meteor-data

npm install --save react react-dom react-addons-pure-render-mixin
```

Then, in your application, you should import React directly rather than relying on a global React symbol:

```
import React from 'react';
```

If you are using a package that depends on the `react` or `react-runtime` Atmosphere packages, you will still need to install the full list of npm React packages above, so we encourage package authors to update their packages to import React directly from npm.

Loading data with React

The `react-meteor-data` has a new `createContainer` syntax for combining Meteor's data system with React in an idiomatic way. We encourage you to use containers to separate your data loading concerns from your presentational components!

Install Angular from npm

With an Angular Meteor app you can safely update to Meteor 1.3 without any changes to your code. You need to make sure you are using the latest `angular` Atmosphere package `1.3.9_2`.

But, in Meteor 1.3, we recommend installing `angular` and `angular-meteor` into your app using npm:

```
npm install --save angular angular-meteor
```

and importing them from your app code:

```
import angular from 'guide/site/content/angular';
import angular

-meteor
from
'angular-meteor';
```

Read the Using Packages article for more details about how to manage your npm dependencies.

If you already using the Atmosphere packages and want to move to the npm packages, you will need to remove the Atmosphere packages first but keep the angular-templates Atmosphere package:

```
meteor remove angular
meteor add angular-templates

npm install --save angular angular-meteor
```

Then, in your application, you should import angular directly rather than relying on global angular:

```
import angular from 'angular';
import angular-meteor from 'angular-meteor';
```

Existing Angular Atmosphere packages

If you are a package author that depends on the `angular:angular` Atmosphere package, you can support both Meteor 1.2 and 1.3 so your users will have an unbreaking update process:

Change your `angular:angular` dependency into a weak dependency:

```
api.use('angular:angular@1.5.3', 'client', { weak: true });
```

and then add a dependency check for both Meteor 1.2 and 1.3 before initializing your angular module:

```
if (!window.angular) {
  try {
    if (Package['modules-runtime']) {
      var require = Package['modules-runtime'].meteorInstall();
      require('angular');
    }
  } catch(e) {
    throw new Error('angular package is missing');
  }
}

angular.module('your.module', []);
```

New guide articles

As part of the 1.3 release, we have some new guide articles and updated sections of existing articles:

- There's a Application Structure article which explains how to structure your files and use the module system.
- There's a Code Style article that makes recommendations about how to ensure consistent formatting for your code.
- There's a Testing article which covers how to do various kinds of testing in Meteor.
- There's a React article which explains how to best use React with Meteor
- There's a Mobile article which covers how to best use our Cordova integration.
- There's a Using Packages article which explains how best to use both npm and Atmosphere packages in your app.
- There's a Writing Packages article which explains practice for writing Atmosphere packages and using all kinds of dependencies within them.
- The UI/UX article has been updated to explain how to do i18n in Meteor applications.