Hot Code Push

Is your Meteor Cordova app not getting the updates you're deploying?

After reading this article, you'll know:

1. The prerequisites to using Hot Code Push
2. Some techniques to diagnose and solve common issues
3. How to dig deeper if that doesn't solve your issue

This article builds on the Cordova article. We recommend reading that first, though we've tried to link back to its relevant sections.

Prerequisites

Make sure that you have:

- an Android and/or iOS mobile app based on Meteor's Cordova integration
- the package `hot-code-push` listed in your `.meteor/versions` file
- locally: make sure your test device and development device are on the same network
- in production: make sure the `--server` flag of your `meteor build` command points to the same place as your `ROOT_URL` environment variable (or, on Galaxy, the *site* in `meteor deploy site`). See details

Known issues

Override compatability versions

Did the app suddenly stop getting new code after you updated meteor, or you changed plugins?

The client probably logs: `Skipping downloading new version because the Cordova platform version or plugin versions have changed and are potentially incompatible`

Meteor, Cordova and plugins cannot be updated through Hot Code Push. So Meteor by default disables Hot Code Push to app versions that have different versions than the server. This avoids crashing a user's app, for example, when new JS calls a plugin that his app version doesn't yet have.

You can override this behavior. Just make sure you deal with potentially incompatible versions in your JS instead.

Update your AUTOUPDATE_VERSION

`AUTOUPDATE_VERSION` is an environment variable you can add to your `run` and `deploy` commands:

```
$ AUTOUPDATE_VERSION=abc meteor deploy example.com
```

If your app has an `AUTOUPDATE_VERSION` set, make sure you change its value when you want a deploy to update your clients.

Cordova doesn't hot reload CSS separately

Are you seeing your web app incorporate changes without reload, yet your cordova app reloads each time?

For CSS-only changes, this is the expected behaviour. Browsers update the layout without reload, but in cordova, any change reloads the whole app.

In case you want to implement soft CSS update for Cordova, see below how to edit the source.

Outdated custom reload code and packages

There are several reload packages, and maybe your app includes some custom reload code. Of course, these may have bugs or be outdated.

In particular, when you push an update, does the app reload but use the old code anyways? Probably, the code hasn't been updated to work with Meteor 1.8.1 or later. As mentioned in the changelog, we recommend you call `WebAppLocalServer.switchToPendingVersion` before forcing a browser reload.

Alternatively, use the built-in behavior to reload. Instead of, say, `window.location.reload()`, call the `retry` function passed to the `Reload._onMigrate()` callback. For example:

```
Reload._onMigrate((retry) => {
  if (/* not ready */) {
    window.setTimeout(retry, 5 * 1000); // Check again in 5 seconds
    return [false];
  }
  // ready
  return [true];
});
```

If you use a package that is no longer compatible, consider forking it or opening a PR with the above changes. Alternatively, you can switch to a compatible one such as `quave:reloader`

Avoid hash fragments

Cordova doesn't show the URL bar, but the user is still on some URL or other, which may have a hash (`#`). HCP works better if it doesn't.

If you can, remove the hash fragment before the reload.

Avoid making it download big files

In the client side logs, you may see HCP fail with errors like:

```
Error: Error downloading asset: /
  at http://localhost:12472/plugins/cordova-plugin-meteor-webapp/www/webapp-local-server.js
  at Object.callbackFromNative (http://localhost:12472/cordova.js:287:58)
  at <anonymous>:1:9
```

This error from cordova-plugin-meteor-webapp may be caused by big files, often in the `public` folder. Downloading these can fail depending on connection speed, and available space on the device.

You could run `$ du -a public | sort -n -r | head -n 20` to find the 20 biggest files and their sizes. Consider serving them from an external storage service or CDN instead. Then they are only downloaded when really needed, and can fail downloading without blocking HCP.

If it is only broken locally

If you notice HCP works in production but not when you test locally, you may need to enable clear text or set a correct `--mobile-server`. Both are explained in the docs.

Still having issues?

If none of that solved your issues and you'd like to dive deeper, here's some tips to get you started.

If you end up finding a bug in one of Meteor's packages or plugins, don't hesitate to open an issue and/or a pull request.

Where does hot code push live?

Hot code push is included in `meteor-base` through a web of official meteor packages, most importantly `reload` and `autoupdate`.

In the case of cordova, a lot of the heavy lifting is done by `cordova-plugin-meteor-webapp`.

To oversimplify, `autoupdate` decides *when* to refresh the client, the plugin then downloads the new client code and assets, and `reload` then refreshes the page to start using them.

What are the steps it takes?

We can break it down a bit more:

- whenever the server thinks the client side may have changed, it calculates a hash of your entire client bundle
- it publishes this hash to all clients
- the clients subscribe to this publish
- when a new hash arrives, each client compares it to its own hash
- if it's different, it starts to download the new client bundle
- when it's done, the client saves any data and announces that it will reload

- the app and packages get a chance to save their data or to deny the reload
- if/when allowed, it reloads

How to spy on it?

To figure out where the issue is, we can log the various steps HCP takes.

First, make sure you can see client-side logs (or print them on some screen of your app).

A few more useful values to print, and events to listen to, might be:

- The version hashes: `__meteor_runtime_config__.autoupdate.versions['web.cordova']`
- The reactive `Autoupdate.newClientAvailable()`: if this turns into `true` and then doesn't refresh, you know the client does receive the new version but something goes wrong trying to download or apply it.

```
Tracker.autorun(() => {
  console.log('new client available:', Autoupdate.newClientAvailable());
});
```

- To check the client's subscription to the new versions, check `Meteor.default_connection._subscriptions`. For example, to log whether the subscription is `ready` and `inactive` (using lodash):

```
const { ready, inactive } = _.chain(Meteor)
  .get('default_connection._subscriptions', {})
  .toPairs()
  .map(1)
  .find({ name: 'meteor_autoupdate_clientVersions' })
  .pick(['inactive', 'ready']) // comment this to see all options
  .value();
console.log('ready:', ready);
console.log('inactive:', inactive);
```

Or, to log the value of `ready` each time the subscription changes:

```
const hcpSub = _.chain(Meteor)
  .get('default_connection._subscriptions', {})
  .toPairs()
  .map(1)
  .find({ name: 'meteor_autoupdate_clientVersions' })
  .value(); // no .pick() this time; return whole subscription object

Tracker.autorun(() => {
  hcpSub.readyDeps.depend(); // Rerun when something changes in the subscription
  console.log('hcpSub.ready', hcpSub.ready);
});
```

Should print `false` and then `true` less than a second later.

- To see if we finish downloading and preparing the new version, listen to `WebAppLocalServer.onNewVersionReady`;

```
WebAppLocalServer.onNewVersionReady(() => {
  console.log('new version is ready!');
  // Copied from original in autoupdate/autoupdate_cordova.js because we overwrite it
  if (Package.reload) {
    Package.reload.Reload._reload();
  }
});
```

- To see if permission to reload is being requested, listen to `Reload._onMigrate()`. Be sure to return `[true]` or the reload may not happen. (I believe that if this is run in your app code, it means all packages allowed the reload. But I didn't find my source on this.)

```
Reload._onMigrate(() => {
  console.log('going to reload now');
  return [true];
});
```

- To know if a run of `Meteor.startup` was the result of a HCP reload or not, we can take advantage of the fact that `Session`s (like `ReactiveDict`s) are preserved.

```
Meteor.startup(() => {
  console.log('Was HCP:', Session.get('wasHCP'));
  Session.set('wasHCP', false);

  Reload._onMigrate(() => {
    Session.set('wasHCP', true);
    return [true];
  });
});
```

How to edit the source

Finally, if you want to change some of the package and plugin code locally, you can.

Editing the packages

Say we want to edit the `autoupdate` package.

In the root of your project, create a folder named `packages`, then add a folder `autoupdate`. Here we put the code from the original package (found in `~/.meteor/packages`), then we edit it.

Meteor will now use the local version instead of the official one.

Editing the plugin

To install a modified version of a plugin,

- from another folder, download the original code e.g. `git clone https://github.com/meteor/cordova-plugin-meteor-webapp.git`
- install it into your meteor project with `meteor add cordova:cordova-plugin-meteor-webapp@file://p`
- modify it as you like

Meteor will start using the local version instead of the official one. But note you will have to rerun `meteor build` or `meteor run` every time you change the plugin.

Found a bug?

If you found a bug in one of the packages or plugins, don't hesitate to open an issue and/or pull request.