interactone
ECOMMERCE PERFORMANCE SPECIALISTS

White Paper

Migrating to Magento 2: What They Don't Tell You

Magento™ PARTNER

513.469.7042 | interactOne.com

Planning

In preparation for Magento 2 (M2), Magento published a series of M2 migration guides to help the community learn how to migrate from Magento 1 (M1) to the new platform. The developer documentation covers everything from best practices, how the migration works, using the data migration tool and testing after the migration.

While the M2 migration guide and plan are very helpful when looking to transfer data from the M1 site to the M2 site, those guides focus on what Magento was able to automate: transferring *data*. Orders, products, customers, promotions, etc can be easily transferred from M1 to M2 without issue, but the same cannot be said about the M1 theme and custom module/extension files. It's important to plan how the entire M1 site (themes, custom modules, media, etc) can be migrated to the M2 site.

Note: Before getting started, evaluate the status of the site and determine what actually needs migrated.

Before looking at *how* individual parts of a M1 site can be migrated to M2, it's best to evaluate the status of the site and determine what actually needs to be migrated. Are all of the installed themes being used? Are all of the third party extensions and custom modules ready? Is the media directory full of outdated or unused files? Are there edits to the core that must be rewritten within M2?

Strip the M1 installation down to the components that are vital for the site to run properly with the desired functionality of the shop owner. There won't be a better time to tidy things up than when planning the M1 to M2 migration.

Theme

In order to migrate a M1 theme to M2, it's important to understand the differences between a M1 theme and a M2 theme. The theme files remain within app/design/frontend, but there are some notable differences between the M1 theme structure and the updated structure of M2 themes. Rather than having typical layout, template and locale directories within app/design/frontend/<package>/<theme>/, M2 has increased the modularity of themes by including ALL theme files within the same directory.

```
<theme>/ (app/design/frontend/package/theme)
├── <Vendor>_<Module>/ (theme-specific module overrides)
│       ├── web/
│       │       ├── css/
│       │       │       ├── source/
│       ├── layout/
│       │       ├── override/
│       ├── templates/
├── etc/ (theme configuration XML files)

│       ├── view.xml (product image configuration*)
├── i18n/ (translations)
├── media/ (screenshot of the theme*)
├── web/ (static files)
│       ├── css/ (theme css)
│       │       ├── source/ (less files)
│       ├── fonts/ (theme fonts)
│       ├── images/ (theme images)
│       ├── js/ (theme javascript)
├── composer.json (theme dependencies and metadata)
├── registration.php (registers the theme*)
├── theme.xml (declares the theme and includes theme metadata*)
* = required
```

Theme-specific module overrides, styles, js, media, etc are among the additions to the new theme structure. Translations have been moved from the /locale/ directory into a new directory, /i18n/. There are also required theme files in M2, which declare and register the theme with the system.

Adding 'theme.xml' within the theme directory will declare the theme with Magento. This file contains the title of the new theme, the parent theme (if there is a parent) as well as the path to the theme's preview image. The preview is typically a screenshot of the theme so it's easy to switch between themes via the admin panel.

Registering the theme is accomplished by adding 'registration.php' within the theme directory. This file is calling a core 'register' method within the Framework module's Component Registrar class. The register method's first parameter dictates that a theme is being registered, the second parameter is the path to the theme after 'app/design' (Ex: 'frontend/Magento/luma') and the third parameter is the directory of the registration file.

M1 themes need to be restructured for M2 and the required files must be added. There are some additional files that can be added for further customization such as making the theme a Composer package, setting the logo (now completed via xml) and setting product image sizes. For additional information, see the 'Themes' section of the M2 developer documentation.

Modules and Extensions

Many M1 extension authors have published M2 versions of their extensions already. If at all possible, updating extensions (especially paid extensions, rather than custom modules) should be left up to the extension authors. If an extension does not have a M2 version yet, reach out to the company and make sure that there will be a M2 version and that it will be published in time for the migration.

While M1 is considered to be modular, the only truly 'modular' part of extensions (or '*modules*') is the code that resides within the 'local' or 'community' directories. Layout XML, templates, JavaScript, CSS, images, etc could be spread throughout the installation in any number of locations depending on the number of themes that are in use. It is impossible to select a single directory and remove an entire module.

Similar to the changes in the theme structure of M2, the module structure is much more modular in M2. This is an improvement, in that sense, because all of the modules are truly self-contained features. This won't mean much to shop owners, but developers will have a much easier time navigating the file system and, ultimately, developing custom functionality.

The structure of the M2 modules is different, so gathering all of the M1 module files and organizing them in the way M2 expects should be enough, right? Not quite. In addition to changes in structure, the core modules themselves have been rewritten. This means that, for the most part, the custom modules will need rewritten.

Note: Check that your extensions have been updated by their authors. They must be updated in order to migrate.

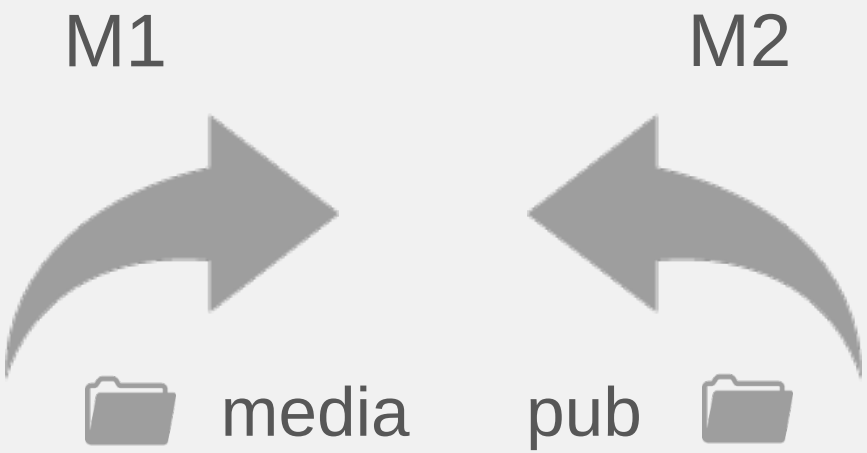
An interesting change in M2 is that almost everything can be accomplished via the API. While the M1 API was helpful when integrating with 3rd party systems, (in *most* cases) there was no real benefit from using API methods as opposed to core methods. M2 is, at its core, API centric. Developers are encouraged to use API methods when developing custom functionality in order to increase extensibility and standardization.

Another major change is the use of the '__construct' method. The construct method is used to load any required classes, set variables, and more depending on what the class is doing. Adding the construct method is required when adjusting a M1 module for M2.

In addition to construct, the 'execute' method is crucial in M2. Each controller only has one action in M2 and that action is 'execute'. While this can increase the number of files in a module, it cleans up controllers that could otherwise reach hundreds (if not thousands) of lines. Security is also improved since an exact class name is required to trigger a controller action rather than having customer or proprietary data accessible via multiple actions within the same class. There are a number of additional changes to the M2 module structure and best practices, which the M2 developer documentation outlines.

Media

Migrating the M1 media files to M2 is fairly straightforward. To transfer media files stored in the M1 file system, simply copy the contents of the '/media' directory into the '/pub/media' directory in the M2 file system.



One caveat is that .htaccess files within the M1 media directory **should not** be directly copied into the M2 file system. M2 is using new .htaccess files that should not be overwritten by legacy .htaccess files.

If the media files are stored in the database, they should be migrated *before* the 'migrate data' step when using the Magento data migration tool. The M2 developer documentation outlines the process of migrating the media files from the M1 database into the M2 database.

Note: .htaccess files within the M1 media directory should NOT be copied directly into the M2 file system.

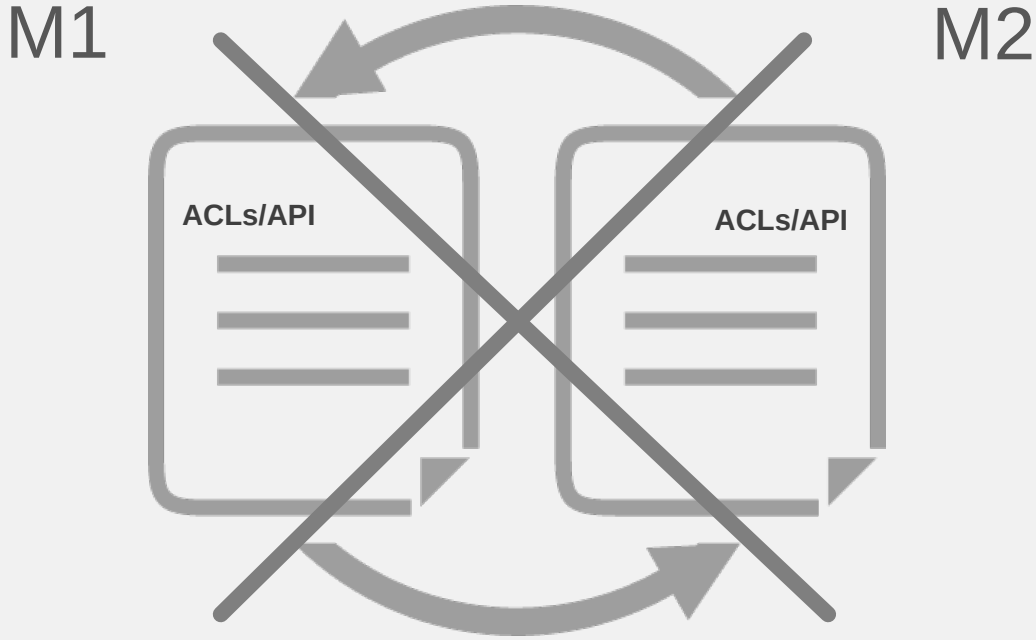
Access Control Lists

Access Control Lists (ACLs) and Application Programming Interface (APIs) data cannot be transferred via the data migration tool. In order for admin users to have access to the new M2 admin, ACLs data must be regenerated for M2. In addition, credentials for web services APIs (SOAP, XML-RPC, REST, etc) must be regenerated for M2 also.

Users will not be able to access the admin using their old M1 credentials or their old 'user groups' since it is not included in the data migration.

It's best to determine where privileges can or should be adjusted early on so the admin processes are secure and each admin has the correct access. Some M1 admin features have changed or moved in the M2 admin.

Note: Admin users and privileges as well as web services will need to be regenerated for M2



ACLs and APIs data cannot be transferred via the data migration tool.

It's pretty safe to say that converting M1 theme and modules to M2 will consume most of the migration time, but the amount of time really depends on the plan set in place upfront.

It's imperative that shop owners and their developers have a strong overall understanding of what exists in M1, what is actually necessary to keep and how to accomplish that in M2.

interactone
ECOMMERCE PERFORMANCE SPECIALISTS

About Us

interactOne has provided web development and Internet marketing services since 1998, becoming a Magento partner in 2009. We have helped many companies develop and establish very successful internet strategies, websites, and eCommerce sites.

Our focus is custom, on-shore development of eCommerce sites built to high standards for usability, speed and delivery of business objectives. We are committed to our clients' results and we plan every site carefully before writing a single line of code. Finding success in eCommerce is never easy, but we have the experience and knowledge to apply in your favor!

Contact Us

513.469.7042
bdwyer@interactone.com
interactOne.com