django-ecstatic Documentation

Release 0.1

Matthew Tretter

July 15, 2013

CONTENTS

CHAPTER

ONE

A QUICK TOUR

django-ecstatic is an expansion pack for `django.contrib.staticfiles`! Read the full documentation at readthedocs.

Here are some things it can do:

- Eliminate the need for network connections to calculate file hashes.
- Collect only the static files that have changed.
- Add content hashes to your filenames, without failing on non-existent files.
- Create static manifests to reduce network operations and ease deployment.

Ecstatic's utlities are written with the same interfaces as `django.contrib.staticfiles`, so they should be compatible with your favorite Django storage libraries.

1.1 Hashed Filenames FTW

First of all, you should already be using Django's CachedFilesMixin or CachedStaticFilesStorage classes, which add a post-processing step to the *collectstatic* command that saves a copy of your static files with a hash of their contents in the filename. If you're serving the files yourself, this will allow you to set far-future expirations for your assets, which will make your site's users happy. It also means that new versions of assets won't overwrite the old versions, which would break your site if the deployed code and static files aren't in sync.

However, in order to get the content hash, these classes will open the file using your app's STATICFILES_STORAGE. If you're using a CDN, this means they'll be performing network operations. But those static files are saved on the local filesystem, too—after all, they were collected from somewhere. That's where `ecstatic.storage.CachedStaticFilesMixin` and `ecstatic.storage.CachedStaticFilesStorage` come in. Instead of using the storage class to get the hash, they'll use your app's staticfiles finders to find the local version and use its hash. (They also have a couple of other handy features.) Use the mixin with the storage class of your choice to get the benefits:

```
from ecstatic.storage import CachedStaticFilesMixin
from cumulus.storage import CloudFilesStaticStorage


class MyStaticFilesStorage(CachedStaticFilesMixin, CloudFilesStaticStorage):
    pass
```

1.2 Hashed Filenames and Built Files

Remember when I mentioned how `ecstatic.storage.CachedStaticFilesMixin` and `ecstatic.storage.CachedStaticFilesStorage` worked? They calculate the hashes of the local versions of the static files. Obviously, then, the local versions—that is, the static files on your app server—need to be the same as the ones you collected to your CDN. Otherwise, the app server would get different hashes and use the wrong URL! So if your project requires a build step, you need to make sure that the built files are on your app server. There are two ways to do this:

1. Include your built files in your package and deploy them with the rest of your application code.
2. Re-build the static files on the app server.

Alternatively, you can go back to using `django.contrib.staticfiles.storage.CachedFilesMixin` or `django.contrib.staticfiles.storage.CachedStaticFilesStorage`, though then you're back in the situation of using network operations to get the hash.

All of the above options have pros and cons. If you deploy directly from version control, option 1 would mean committing compiled files to your repository, which you may consider bloat. On the other hand, option 2 means that your app server needs to have all of your build tools installed. It also means that there will be some time while new code is deployed, but it's referencing old assets (until the build completes so the storage can get the new hash).

Luckily, Ecstatic has another solution: `ecstatic.storage.StaticManifestMixin`. This mixin is used just like `ecstatic.storage.CachedStaticFilesMixin`, but it looks up your static files URLs in a manifest file—completely sidestepping the need to calculate the hash of the local files.

```
from ecstatic.storage import CachedStaticFilesMixin, StaticManifestMixin
from cumulus.storage import CloudFilesStaticStorage


class MyStaticFilesStorage(StaticManifestMixin, CachedStaticFilesMixin, CloudFilesStaticStorage):
    pass
```

Note: Notice that we're still including `CachedStaticFilesMixin`. It (or Django's version) is still needed for the post-processing, and to figure out which URL should be inserted into the manifest.

With this mixin, the storage no longer needs access to the built files to determine their hashes (and therefore URLs); it only needs to access the manifest file. That means:

- You don't need to package your built static files with your app.
- You don't need to install your build tools on your app server.
- The storage class can lookup the new URL as soon as new code is deployed (along with a new staticfiles manifest).
- We still aren't performing network operations to get hashes/URLs.

In other words, we've solved all of our issues. Yay!

So how do you create this manifest? First, you need to add a variable to your settings.py file to let Ecstatic know where to create it:

```
ECSTATIC_MANIFEST_FILE = os.path.join(os.path.dirname(__file__), 'staticmanifest.json')
```

Then just run the `createstaticmanifest` management command:

```
./manage.py createstaticmanifest
```

Note: When you run `createstaticmanifest`, make sure that the Django settings you're using contain the correct `STATICFILES_STORAGE`. If you have a local_settings.py that sets a different `STATICFILES_STORAGE`, the manifest will contain the URLs that it reports!

CHAPTER

TWO

CONTENTS

2.1 Settings

2.1.1 Storage Settings

django.conf.settings.**ECSTATIC_STRICT**

Default `False`

Specifies whether Ecstatic should raise an exception when it can't convert a URL. If `False`, the unconverted URL will be used. This situation typically arises when a CSS file points to a non-existent image.

2.1.2 Manifest Settings

django.conf.settings.**ECSTATIC_MANIFEST**

Default `'ecstatic.manifests.JsonManifest'`

The dotted path to the manifest class to be used by the `createstaticmanifest` management command and `ecstatic.storage.StaticManifestMixin`.

django.conf.settings.**ECSTATIC_MANIFEST_FILE**

Default `None`

The path where the manifest file should be saved. This setting must be set if using staticfiles manifests.

django.conf.settings.**ECSTATIC_USE_MANIFEST**

Default the opposite of `DEBUG`

Specifies whether the manifest should be used by the (`ecstatic.storage.StaticManifestMixin` extending) storage class.

django.conf.settings.**ECSTATIC_MANIFEST_CACHE**

Default `'ecstatic_manifest'`, if the `CACHES` dictionary contains it,

otherwise 'default'

The name of the cache that should be used by the manifest class.

django.conf.settings.**ECSTATIC_MANIFEST_EXTRAS**

Default `['admin/']`

A list of paths that will not be found by Django's `STATICFILES_FINDERS` but that should nonetheless be included in the manifest. This setting exists chiefly to support the Django admin, which uses the static template tag with the path `'admin/'` in order to get a static prefix for the admin. (Note that Django's behavior here may be incompatible with storages that alter the filepath in a way other than adding a prefix.)

2.2 Storages

2.3 Management Commands

2.3.1 collectnewstatic

2.3.2 createstaticmanifest

2.3.3 hashmedianames

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*