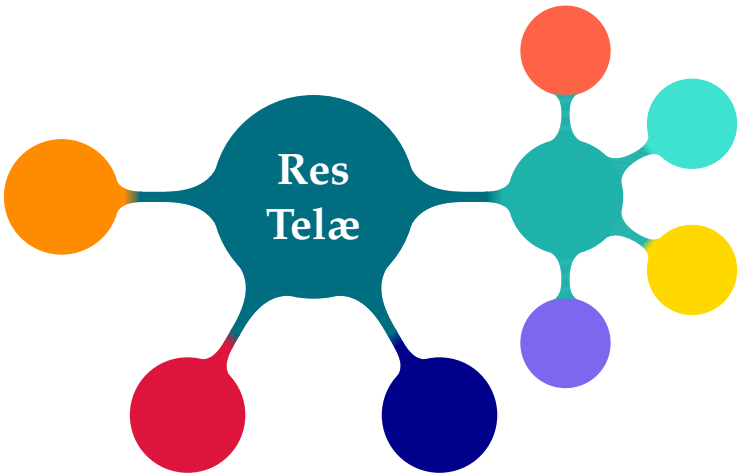
Migration de données sous Drupal

Felip Manyer i Ballester



16 avril 2014

À propos de moi

- Drupalien depuis fin 2009, centralien de Lyon (2008).
- Exerce en indépendant à Perpignan sous le nom commercial « Res Telæ ».
- Vient aux meetups pour donner libre cours à sa logorrhée, mais aussi rencontrer des gens comme lui.
- « Anarchiste » approchant dangereusement la trentaine (J-6), fermement opposé à la confiscation et la minitellisation d'Internet.
- Loisirs : natation, cyclisme, sports de montagne, piano, sciences naturelles, linguistique, OpenStreetMap, Guifi (?), changer le monde...

Plan

Migration de données : pourquoi ?

Récupération de données

- Depuis une ancienne version du site (autre technologie) ;
- depuis une ancienne version majeure de Drupal ;
- depuis n'importe quelle source structurée (SGBDR, flux ou fichiers XML, JSON, CSV, etc.) ;

→ une seule fois ou régulièrement.

Exportation de données

- On en parle brièvement en partie 4 ;
- voir aussi présentation d'Alexandre.

Pourquoi importer dans Drupal ?

Lecture à la volée de la source externe

- Envisageable dans certains cas (`db_select()`).
- Pas toujours fiable (cf. Simon).
- Ressources gaspillées.
- Interaction limitée avec les outils Drupal.

Importation sous forme d'entités natives

- Résout problèmes ci-dessus.
- Exploitation des données de manière drupalienne : *entités* (nœuds, taxonomie, personnalisées...), références entre entités, *view modes*, *Views*, *EntityFieldQuery*, recherche, etc.

Concepts généraux

Source Données extérieures à importer dans Drupal (autre CMS, fichiers structurés, flux, base de données).

Destination Nouveau support destiné à accueillir les données (nœud, utilisateur, etc.)

Correspondance Description des relations entre éléments de la source et de la destination. Exemple : « la balise RSS `pubDate` correspond au champ `date` de mon type de contenu `evenement` ».

Méthodologies envisageables

Développement (personnalisé) méthode historique, toujours valable.

Aggregate (module du cœur) importation de flux XML (RSS, Atom, RDF). Fonctionnalités et intégration avec le reste de l'écosystème assez vite limitées.

Feeds outil orienté « site builder ». Définition des importations (par défaut : nœuds et utilisateurs) via l'interface. Répond à de nombreux cas d'utilisation.

Migrate orienté développeur. Écriture de classes PHP. Plein contrôle sur le pré-traitement des données. Interfaces : UI, Drush.

Utilisation de Feeds

Création d'un « importer »

Choisir et configurer :

Fetcher mode de récupération des données (par défaut, fichier ou via HTTP) ;

Parser grammaire selon laquelle les données sont présentées (RSS, CSV, etc.) ;

Processor choix de l'entité de destination (par défaut : nœud, terme de taxonomie, utilisateur) + mise en correspondance (« mapping ») avec champs et propriétés.

☞ Démonstration

Feeds : quelques caractéristiques

+ Gestion native des importations périodiques ;
+ importateurs exportables (!) en code avec Features ;
+ support d'XPath ;
~ support limité du pré-traitement des champs avec Feeds Tamper ;
– pas de traitement conditionnel ;
– support déficient du multilinguisme.

Migrate

Migrate offre un cadre de développement orienté objet permettant d'effectuer des migrations de données y compris dans des situations complexes.

Module en lui-même réservé aux développeurs, mais :

- utilisable par tous → interface utilisateur puissante offerte par ce module (analyse, lancement des importations, progression, arrêt, retour en arrière (rollback)) ;
- pilotable avec Drush ;
- il existe des modules basés sur Migrate répondant à des cas d'utilisation précis : migration depuis Wordpress, phpBB, un autre Drupal...

Architecture générale de Migrate

Par convention, les différents éléments sont à placer dans `MODULE.migrate.inc`, il faut y implémenter :

- `hook_migrate_api()` ;
- une classe héritant de `Migration`, qu'il faut enregistrer, comprenant parmi ses propriétés 4 objets instanciant les classes `MigrateSource`, `MigrateDestination`, `MigrateMap` et `MigrateFieldMapping`.

Migrate : les quatre classes

MigrateSource	description des champs de données à la source ;
MigrateDestination	description des champs de données à la destination ;
MigrateFieldMapping	correspondance des champs entre source et destination ;
MigrateMap	garde trace des champs de la source (et leurs types) dont découle un objet dans la destination, utile pour le *rollback*.

Migrate : quelques caractéristiques

- gestion des importations périodiques : `highwater` permet de ne réimporter une entité qu'en cas de changement ;
- pré-traitement des champs avancés grâce à la méthode `prepareRow()` (fusion de champs, abandon d'enregistrements, etc.) ;
- documentation fournie ;
- embarqué dans le cœur de Drupal 8.

☞ Démonstration

Export de données

UUID (Universally Unique IDentifier) exporter du contenu comme du code. À réserver à certains cas limites.

Deployment Voir présentation d'Alexandre.

Views Data Export Format de vue permettant d'exporter les données sous forme de fichiers CSV, Excel, XML ou texte. Intégration avec Drush, et donc scriptable.

Flux de données sérialisées (XML, JSON, etc.), par exemple avec Views, Services web (mais bootstrap) → c'est un autre sujet...

Conclusion

- Des outils pour gérer les importations de données, au sein de systèmes d'information toujours plus complexes ;
- Feeds : permet dans de nombreux cas aux *site builders* de gérer eux-même les importations (mais pourquoi toujours en alpha ?) ;
- Migrate : un socle rigoureux, robuste et puissant, tant pour le développeur que pour l'utilisateur. Standard *de facto* dans l'écosystème (passage d'une version majeure à une autre, D6 $\rightarrow$ D8 et D7 $\rightarrow$ D8) ;
- importation vers tout type d'entité Drupal, y compris les vôtres (cf. juin).