Les générateurs de site Web statiques, et mon choix de Pelican

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 19 octobre 2018

https://www.bortzmeyer.org/generateurs-web-statiques.html

J'ai récemment dû faire deux sites Web et leur cahier des charges permettait de faire des sites Web statiques, ce qui a de nombreux avantages. Parmi les dizaines de logiciels qui permettent de faire un site Web statique, j'ai choisi Pelican <https://getpelican.com/>.

Un de ces deux sites (l'autre est privé) est celui de mon livre <https://cyberstructure.fr/>.

Pourquoi un site Web statique, et non pas généré à la demande via un CMS ? Cela présente plusieurs avantages :

— Le principal est de performance et de résistance à la charge : un site Web dépendant d'un CMS doit exécuter des milliers de lignes de PHP ou de Java à chaque requête. Cela le rend lent et, surtout, une attaque par déni de service, même modérée (par exemple avec un outil primitif comme LOIC) suffit à le rendre inutilisable. Même pas besoin d'attaque, d'ailleurs, parfois, une simple utilisation un peu intense (une mention sur Mastodon <https://github.com/tootsuite/mastodon/issues/4486>, par exemple) peut le rendre inaccessible. Un site Web dynamique doit donc être hébergé sur de grosses machines coûteuses, alors qu'un site statique peut tenir une charge bien plus élevée. Tous les serveurs HTTP savent servir des fichiers statiques très rapidement.
— Un deuxième avantage est de dépendance. Le site statique ne dépend que du serveur HTTP. Au contraire, un site dynamique dépend de plusieurs logiciels supplémentaires (comme par exemple MariaDB) dont la panne peut le rendre inutilisable, souvent avec des messages d'erreur incompréhensibles.
— Enfin, le site statique présente un gros avantage de sécurité : au contraire du site dynamique, qui peut souvent être piraté, par exemple par injection SQL <https://www.bortzmeyer.org/sql-injection.html>, le site statique n'exécutant pas de code et étant composé de simples fichiers de données, n'a pas de failles de sécurité, autres que celles du serveur HTTP et de la machine sous-jacente.

Bref, pas question d'utiliser WordPress ou Drupal. Mais, alors, comment faire ? Une solution évidente est de tout faire à la main, éditant le HTML du site avec un éditeur texte ordinaire. Cette solution n'est pas forcément agréable. Éditer du HTML n'est pas très amusant (même si, au début du Web, tout le monde faisait comme cela car il n'y avait pas le choix), et surtout, cela rend très difficile le maintien d'une apparence cohérente entre toutes les pages du site. Bien sûr, l'utilisation d'un fichier CSS identique pour toutes les pages fera que des questions esthétiques comme la couleur du fond ou comme la police utilisée seront traitées de manière identique pour toutes les pages. Mais cela ne règle pas le cas d'éléments qu'on veut voir partout, et que CSS ne sait pas gérer, ou en tout cas pas facilement avec les navigateurs actuels. Par exemple, si on veut un pied de page identique partout, avec une adresse du webmestre et un menu, faire les pages à la main nécessiterait du copier/coller pénible et, surtout, rendrait difficile toute modification ultérieure de ce pied de page.

Une solution possible serait d'utiliser un CMS pour créer du contenu, ce qui permettrait aux utilisateurs attachés à ces outils de rédiger du contenu, puis de transformer le site Web dynamique en statique, avec un outil comme httrack <https://www.bortzmeyer.org/generer-version-statique-site-web.html>, et de faire servir ensuite le contenu statique. Mais, ici, comme j'étais le seul à ajouter du contenu, le problème d'utilisation d'un éditeur ne se posait pas, je suis plus à l'aise avec un éditeur de texte qu'avec les éditeurs des CMS.

Les générateurs de site Web statiques utilisent tous le concept de gabarit. On écrit un gabarit que doivent respecter les pages, puis on écrit les pages et le générateur de site Web statique réalise l'incarnation de ces pages en fichiers HTML qui suivent le gabarit. De tels outils sont presque aussi anciens que le Web. Le premier que j'ai utilisé était wml <https://www.bortzmeyer.org/wml.html>, puis Blosxom a été le plus à la mode, puis Jekyll. Et il en existe aujourd'hui une quantité étonnante, beaucoup de programmeurs ayant apparemment eu envie d'écrire le leur. Pour en choisir un, mon cahier des charges était logiciel libre, et disponible sous forme d'un paquetage dans la version stable de Debian (car je souhaitais réaliser des sites Web, pas passer du temps à de l'administration système et de l'installation de logiciels).

Voyons maintenant les différents générateurs de sites Web statiques que j'ai testé, en commençant par celui que j'ai finalement choisi, Pelican <http://getpelican.com/>. Pelican est écrit en Python (cela peut avoir son importance si vous souhaitez en étendre les fonctions, et même le fichier de configuration d'un site est en Python). Il est surtout prévu pour des blogs, avec une notion de chronologie, alors que mon utilisation est différente, juste un ensemble de pages. La plupart de ces générateurs de sites statiques sont, comme Pelican, plutôt orientés vers le blog.

Une fois installé (paquetage Debian `pelican`), on crée un répertoire pour les fichiers de son site, on se rend dans ce répertoire, et on lance la commande `pelican-quickstart` qui va interactivement vous guider et générer les fichiers nécessaires. Comme toutes les commandes interactives, c'est pénible à utiliser mais cela peut être utile aux débutants. La plupart des questions sont simples et la valeur par défaut est raisonnable, sauf pour les dernières, où `pelican-quickstart` vous demande comment installer les pages produites sur votre site Web. Il propose FTP, SSH, et des commandes spécifiques pour des environnements fermés chez des GAFA comme Dropbox ou S3. Personnellement, je n'aime pas que la réponse à la question « *"Do you want to specify a URL prefix?"* » soit Oui par défaut. Dans ce cas, Pelican définit une variable `SITEURL` dans la configuration et tous les liens sont préfixés par cet URL, ce qui empêche de tester le site en local, un comble pour un générateur de sites statiques. Je réponds donc Non à cette question (ou bien je mets `RELATIVE_URLS` à `True` dans `publishconf.py`).

Ensuite, nous pouvons rédiger le premier article. Pelican accepte des articles faits en Markdown et reST. Ici, j'utilise Markdown. J'édite un premier article `content/bidon.md`, et j'y mets :

```
Title: C'est trop tôt
Date: 2018-10-11

Test *bidon*, vraiment.
```

Les deux premières lignes sont des métadonnées spécifiques à Pelican. Il faut indiquer le titre de l'article et la date (rappelez-vous que Pelican est optimisé pour des blogs, où les articles sont classés par ordre rétro-chronologique). Si, comme moi, vous faites un site Web qui n'est pas un blog, la date n'est pas nécessaire (les détails suivent). Ensuite, `pelican-quickstart` ayant créé le Makefile qui va bien, un simple `make publish` suffit à fabriquer les fichiers HTML, dans le répertoire `./output`. On peut alors pointer son navigateur Web favori vers `output/index.html`. (`pelican content` donne le même résultat que `make publish`. Tapez `make` sans argument pour voir la liste des possibilités.) Pour copier les fichiers vers la destination, si vous avez configuré un des mécanismes de téléversement, vous pouvez faire `make XXX_upload`, où XXX est votre méthode de téléversement. Par exemple, si vous avez configuré SSH, ce sera `make ssh_upload` (mais vous pouvez évidemment éditer le Makefile pour donner un autre nom).

Si vous voulez changer la configuration du site après, les fichiers de configuration sont eux-même écrits en Python, donc il est utile de connaitre un peu ce langage pour éditer `pelicanconf.py` et `publishconf.py`. Par exemple, si vous voulez que les dates soient affichées en français, avoir répondu `fr` aux questions de `pelican-quickstart` ne suffit pas, il faut ajouter `LOCALE = ('fr_FR', 'fr')` à `pelicanconf.py`.

En mode blog, Pelican met chaque article dans une catégorie, indiquée dans les métadonnées (si vous ne mettez rien, comme dans l'exemple de fichier Markdown plus haut, c'est mis dans la catégorie « *"misc"* »).

Je l'avais dit, les deux sites Web où j'utilise Pelican ne sont pas de type blog, mais plutôt des sites classiques, avec un ensemble de pages, sans notion d'ordre chronologique. Pour cela, il faut mettre ces pages (Pelican appelle les fichiers d'un blog chronologique « articles » et les autres fichiers, par exemple les mentions légales, des « pages ») dans un répertoire `content/pages`, et dire à Pelican sous quel nom enregistrer l'HTML produit, sinon, ses choix peuvent être différents de ce qu'on aurait choisi (mais on n'a plus à mettre la date) :

```
% mkdir content/pages
% emacs content/pages/mentions-legales.md
% make publish
Done: Processed 3 articles, 0 drafts, 1 page and 0 hidden pages in 0.12 seconds.
```

Vous voyez que vous avez maintenant « *"1 page"* », en plus des articles. Ces pages apparaitront en haut de la page d'accueil (tout ceci peut évidemment se changer, voyez plus loin quand on parlera des thèmes).

Pour avoir les pages enregistrées sous un nom de fichier de son choix, on utilise `save_as` :

```
% cat content/pages/mentions-legales.md
Title: Mentions légales
save_as: legal.html

Faites ce que vous voulez avec ce site.
```

(Voir la bonne FAQ <http://docs.getpelican.com/en/stable/faq.html#how-can-i-override-the sur ce sujet.) Évidemment, dans ce cas, le menu automatique ne marchera plus et il faudra, dans le thème qu'on développe, mettre son menu à soi. Mais je n'ai pas encore expliqué les thèmes. Avant cela, notons qu'on peut redéfinir la page d'accueil de la même façon <http://docs.getpelican.com/en/stable/faq.html#how-can-i-use-a-static-page-as-my-home-page>, avec `save_as: index.html`.

Alors, les thèmes, c'est quoi ? C'est un ensemble de gabarits HTML, de fichiers CSS et d'images qui, ensemble, vont assurer la cohérence du site Web, ses menus, son apparence graphique. Si vous avez fait votre propre site en suivant les instructions ci-dessus, vous avez utilisé le thème par défaut, qui se nomme « "*simple*" » et qui est livré avec Pelican. C'est lui qui définit par exemple le pied de page « "*Proudly powered by Pelican, which takes great advantage of Python*" ». S'il ne vous plait pas, vous pouvez avoir votre propre thème, soit en l'écrivant en partant de zéro (c'est bien documenté <http://docs.getpelican.com/en/stable/themes.html>) soit en utilisant un thème existant (on en trouve en ligne, par exemple en <http://pelicanthemes.com/>). Et, bien sûr, vous pouvez aussi prendre un thème existant et le modifier. Comme exemple, je vais faire un thème ultra-simple en partant de zéro.

Appelons-le « "*red*" » :

```
% mkdir -p themes/red
% mkdir -p themes/red/static/css
% mkdir -p themes/red/templates
% emacs themes/red/templates/base.html
% emacs themes/red/templates/page.html
% emacs themes/red/templates/article.html
```

Le code de `base.html` est disponible dans ce fichier (en ligne sur `https://www.bortzmeyer.org/files/pelican-red-base.html`), celui de `page.html` dans celui-ci (en ligne sur `https://www.bortzmeyer.org/files/pelican-red-page.html`) et enfin celui de `article.html` à cet endroit (en ligne sur `https://www.bortzmeyer.org/files/pelican-red-article.html`). `base.html` définit les éléments communs aux articles et aux pages. Ainsi, il contient `<footer>Mon joli site.</footer>`, qui spécifie le pied qu'on trouvera dans tous les fichiers HTML.

Pelican permet également d'utiliser des variables et des tests, avec le moteur de gabarit Jinja. C'est ainsi qu'on spécifie le titre d'un article : `{{ SITENAME }} - {{ article.title }}` indique que le titre comprend d'abord le nom du site Web, puis un tiret, puis la variable `article.titre`, obtenue via la métadonnée `Title:` mise plus haut dans le source Markdown.

Une fois le thème créé, on peut l'utiliser en le spécifiant sur la ligne de commande :

```
% pelican -t themes/red content
```

Ou bien si on veut utiliser `make publish` comme avant, on ajoute juste dans `pelicanconf.py` :

```
THEME = 'themes/red'
```

Notez qu'on n'est pas obligé de mettre le thème dans un sous-dossier de `themes`. Si on ne compte pas changer de thème de temps en temps, on peut aussi placer `static` et `templates` dans le dossier `themes`, voire directement à la racine du répertoire de développement.

On a rarement un thème qui marche du premier coup, il y a plusieurs essais, donc un truc utile est la commande `make regenerate`, qui tourne en permanence, surveille les fichiers, et regénère automatiquement et immédiatement les fichiers HTML en cas de changement. Une autre commande utile est `make serve` qui lance un serveur Web servant le contenu créé, le site étant désormais accessible en `http://localhost:8000/` (personnellement, `x-www-browser output/index.html` me suffit).

Si les fonctions de base ne suffisent pas, Pelican peut ensuite être étendu de plusieurs façons différentes. Notez d'abord que Jinja fournit beaucoup de possibilités (cf. sa documentation <http://jinja.pocoo.org/docs/2.10/templates/>). Ensuite, imaginons par exemple qu'on veuille inclure des données CSV dans ses articles ou pages. On peut écrire un "*plugin*" (c'est bien documenté <http://docs.getpelican.com/en/stable/plugins.html>) mais il existe aussi plein de plugins tout faits <https://github.com/getpelican/pelican-plugins>, n'hésitez pas à chercher d'abord si l'un d'eux vous convient (ici, peut-être "*Load CSV*"). Mais on peut aussi envisager un programme dans le langage de son choix, qui lise le CSV et produise le Markdown. Ou bien, puisque les fichiers de configuration sont du Python, on peut y mettre du code Python, produisant des variables qu'on utilisera ensuite dans son thème (notez qu'on peut définir ses propres métadonnées).

Dernière solution, étendre Markdown avec une extension au paquetage Python `markdown`, qu'utilise Pelican. Ces extensions sont bien documentées <https://python-markdown.github.io/extensions/>, et il existe un tutoriel <https://python-markdown.github.io/extensions/api/>. On peut alors programmer dans l'extension tout ce qu'on veut, et produire le HTML résultant, avec Element Tree <https://docs.python.org/3/library/xml.etree.elementtree.html>. C'est personnellement ce que j'utilise le plus souvent.

Pour terminer avec Pelican, quelques bonnes lectures :

— Je rappelle que la documentation officielle <http://docs.getpelican.com/en/stable/content.html> est très bien faite (quoique manquant d'exemples),
— Un bon récit d'une utilisation de Pelican <https://www.fullstackpython.com/blog/generating-static-html> (avec création de thème),
— Un autre récit <http://nafiulis.me/making-a-static-blog-with-pelican.html>, avec une bonne liste d'avantages des sites statiques,
— Et un dernier retour d'expérience <https://www.bassi.io/articles/2014/07/18/building-a-website>,
— Un exemple amusant d'une extension Markdown <https://bytefish.de/blog/markdown_emoji_extension/>,
— Une liste de sites utilisant Pelican <https://github.com/getpelican/pelican/wiki/Powered-by-Pelican>, et les sources de ces sites.

Voici pour Pelican. Et les autres générateurs de sites statiques, du moins les quelques-uns que j'ai testés, parmi la grande variété disponible ? Actuellement, Hugo <https://gohugo.io/> semble le plus à la mode. Il est programmé en Go. Comme Pelican, il est très nettement orienté vers la réalisation de blogs chronologiques. Le paquetage Debian est très ancien par rapport à la syntaxe toujours changeante, et je n'ai pas réussi à traiter un seul des thèmes existants, n'obtenant que des messages incompréhensibles. Je ne suis pas allé plus loin, je ne sais notamment pas si Hugo peut être utilisé pour des sites qui sont des ensembles de pages plutôt que des blogs.

Au contraire d'Hugo, tout beau et tout neuf, Jekyll <https://jekyllrb.com/> est stable et fait du bon travail. Un coup de `jekyll build` et on a son site dans le répertoire `_site`. On peut aussi faire un site non-blog <https://jekyllrb.com/docs/pages/>. Jekyll est lui, écrit en Ruby. Par contre,

je n'ai pas bien compris comment on créait son thème, et c'est une des raisons pour lesquelles j'ai choisi Pelican.

Au contraire de Jekyll, Gutenberg <https://www.getgutenberg.io/> est récent, et développé dans un langage récent, Rust. Trop nouveau pour moi, il nécessite le gestionnaire de paquetages Cargo qui n'existe apparemment pas sur Debian stable.

Écrit dans un langage bien moins commun, Hakyll <https://jaspervdj.be/hakyll/> est en Haskell, et même le fichier de configuration est un programme Haskell. Mais il est inutilisable sur Debian stable, en raison d'une bogue <http://bugs.debian.org/900783> qui affiche un message très clair :

```
AesonException "Error in $.packages.cassava.constraints.flags: failed
to parse field packages: failed to parse field constraints: failed to
parse field flags: Invalid flag name: \"bytestring--lt-0_10_4\""
```

Enfin, j'ai regardé du côté de PyBlosxom <http://pyblosxom.github.io/> mais j'ai renoncé, trouvant la documentation peu claire pour moi.

Les solutions les plus simples étant souvent les meilleures, il faut se rappeler qu'on peut tout simplement faire un site statique avec pandoc. pandoc prend des fichiers Markdown (et bien d'autres formats) et un coup de `pandoc --standalone --to html5 -o index.html index.md` (qu'on peut automatiser avec make) produit de l'HTML. Si on veut donner une apparence cohérente à ses pages Web, pandoc a égakement un système de gabarits (mais, la plupart du temps, il n'est même pas nécessaire, utiliser le gabarit par défaut et définir quelques variables sur la ligne de commande de pandoc peut suffire).

Comme je l'ai dit au début, il y a vraiment beaucoup de générateurs de sites statiques et je suis très loin de les avoir tous testés. Rien que dans les paquetages Debian, il y a encore staticsite <https://github.com/spanezz/staticsite> et blogofile <http://www.blogofile.com/>.

Vu le grand nombre de générateurs de sites Web statiques, on ne s'étonne pas qu'il existe plusieurs articles de comparaison :

— « "*Comparisons with other static site generators*" <https://github.com/Keats/gutenberg#comparisons-with-other-static-site-generators> » est une comparaison de Gutenberg avec d'autres générateurs,
— « "*StaticGen*" <https://www.staticgen.com/> » est très détaillé, avec beaucoup d'informations factuelles,
— « "*HTML and Static site generators*" <http://www.fabacademy.org/archives/2015/eu/labs/cascina/classes/pre-02/> » est plus synthétique, et plus adapté à celui ou celle qui a un projet concret, le précédent article étant plutôt pour celle ou celui qui veut tout apprendre.

Et, évidemment, il existe toujours la solution bien "*geek*" consistant à développer son propre outil, comme je l'avais fait pour ce blog <https://www.bortzmeyer.org/blog-implementation.html>.