



# Formation Drupal 8 Performance & Sécurité

Animée par Romain JARRAUD

# Stagiaires et formateur

- **Stagiaires**

- Nom et profil ?
- Comment avez-vous découvert Drupal ?
- Qu'attendez-vous de cette formation ?

- **Formateur Romain JARRAUD**

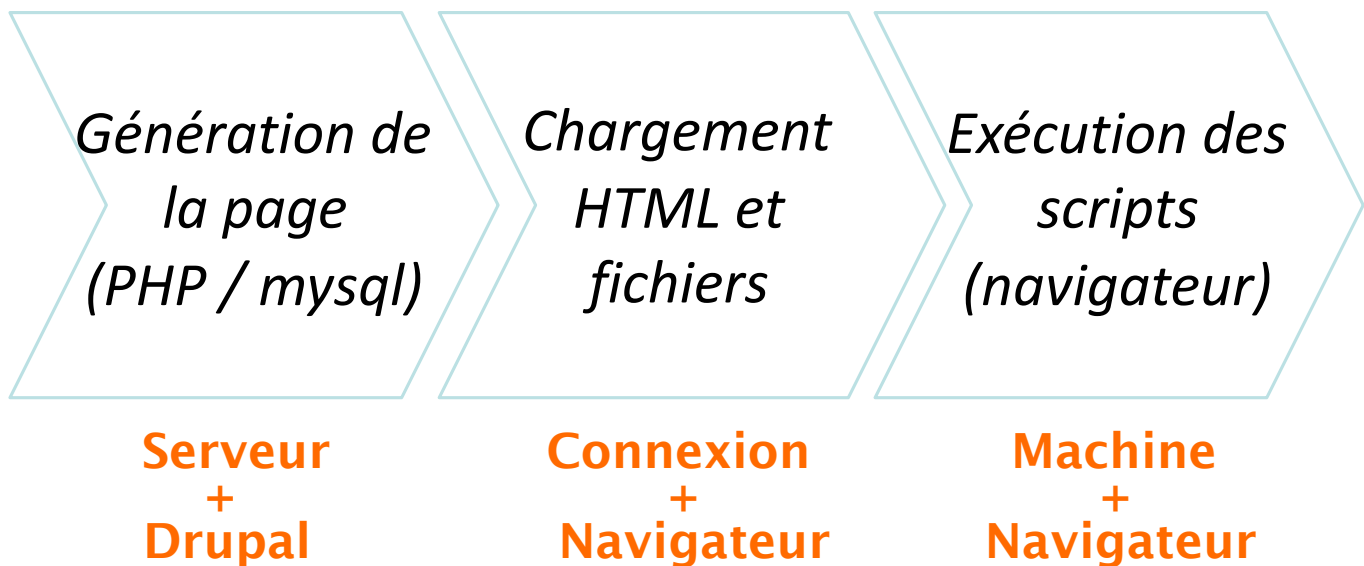
- **Développeur web** depuis 1998.
- A commencé par faire du **développement Drupal**, et aujourd'hui fait de **l'animation de formations et du consulting Drupal**.
- Contact : [romain.jarraud@trainedpeople.com](mailto:romain.jarraud@trainedpeople.com).

# Drupal et la performance

# Objectifs de la formation

- **Comprendre** pourquoi un site Drupal peut être (très) lent.
- **Apprendre** à utiliser les différents modules de cache.
- **Mesurer** la réactivité d'une installation *Drupal*.
- **Améliorer** la configuration du site afin d'obtenir les meilleures performances possibles.

# Performance front vs. serveur



Temps, budget, fonctionnalités... ne pas se tromper de performance !

# Drupal et la performance

- **Drupal** est un système intrinsèquement lourd. Chaque module installé réalise des opérations qui peuvent être consommatrices de ressources. On a de nombreux fichiers PHP impliqués et un grand nombre de requêtes en base de données.
- Le cache natif de **Drupal** est à 100% stocké en base de données. Cette dernière n'est pas faite pour héberger le cache! De fait elle est souvent le goulot d'étranglement.
- De nombreuses couches de cache existent. Comprendre comment elles interviennent est essentiel.
- **Drupal** peut être optimisé « facilement » pour les utilisateurs anonymes, comme pour les utilisateurs authentifiés.



# Optimiser le cache

# Cache

- On peut dissocier principalement 2 types de cache : interne (système) et HTML (rendu). Ce dernier est souvent le plus coûteux en terme de performance car il nécessite pas mal de ressources.
- Activation du module de cache anonyme (***Internal Page Cache***)
- Activation du module de cache dynamique (***Internal Dynamic Page Cache***).



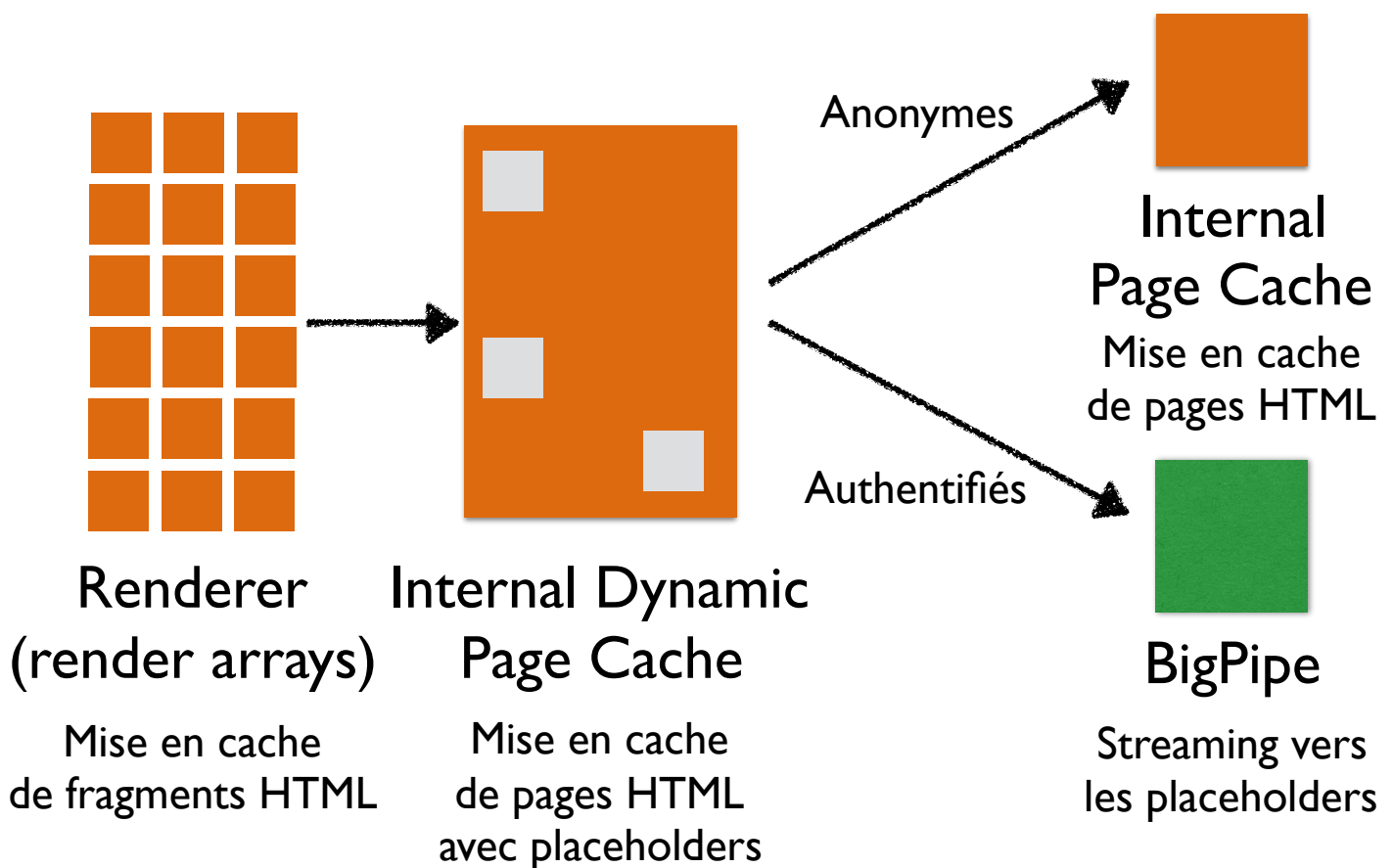
# Les tables de cache

- **cachetags** : ne contient pas des données de caches mais les tags. Ne jamais vider !
- **cache\_bootstrap** : interne (données nécessaires au bootstrap : hook, routing, liste des modules et thèmes...).
- **cache\_config** : interne (stockage des configurations).
- **cache\_container** : interne (mise en cache des services, event dispatcher...).
- **cache\_data** : interne mise en cache données de compression CSS / JS.
- **cache\_default** : mixte (table par défaut).
- **Cache\_discovery** : interne (déclaration dans le code).
- **cache\_dynamic\_page\_cache** : HTML (cache HTML avec placeholder).
- **cache\_entity** : interne (données des entités).
- **cache\_menu** : interne (données de routing).
- **cache\_page** : cache des page pour les anonymes.
- **cache\_render** : fragments HTML de rendu.
- **cache\_toolbar** : toolbar.

# Les modules de cache

- Pour tous les utilisateurs le module **Dynamic Internal Page Cache** permet de mettre en cache une page incomplète (placeholders). La page peut contenir des parties variables en fonction des utilisateurs. Seules ces sections sont reconstruites. Le cache est contextualisé (contexts).
- Pour les anonymes on dispose du module **Internal Page Cache** : la page complète est mise en cache avec une seule variante invalidée éventuellement si elle expose des données qui ont été modifiées depuis le dernier affichage (tags).
- Afin d'améliorer les performances perçues, on dispose du module **Bigpipe**. En utilisant ce qui précède, la page peut être affichée dans le navigateur alors que toutes ses parties n'ont pas été encore calculées. Elles alimentent des placeholders dans le squelette HTML. Le cache est contextualisé (contexts). La page est mise en cache avec ses placeholders avant leurs substitutions.
- Il est possible d'installer également le module contrib **Sessionless BigPipe**.

# Les modules de cache



# Cache API

- Chaque render array doit définir son propre cache identifié par des cache **keys**.
- On dispose de 3 paramètres pour gérer la validité du cache :
  - le **contexte** : le rendu dépend de paramètres tels que l'utilisateur connecté, ou la langue du site.
  - les **tags** : ce que l'on affiche dépend d'autres entités du système ou de la configuration. Par exemple lorsque l'on affiche le nom de l'auteur du noeud, le rendu de ce dernier doit être invalidé si l'utilisateur en question modifie son nom.
  - la **durée** : arbitraire, on invalide le cache au bout d'un certain temps.
- Par défaut, un bloc est mis en cache de façon permanente (*max-age: Cache::PERMANENT*) et doit donc définir ses **keys**.

# Cache API

- Cache **context** : ex. 'timezone', 'session' (liste des contextes possibles sur [drupal.org/developing/api/8/cache/contexts](http://drupal.org/developing/api/8/cache/contexts)).
- Cache **tags** :
  - entity (ex. : 'node:4', 'node\_list').
  - config (ex. : 'config:filter.format.basic\_html').
  - custom.
- Cache **max-age** : nombre de secondes minimum avant invalidation. Une valeur de 0 indique qu'il n'y a aucune mise en cache.

```
$build_1 = [  
  '#markup' => $markup,  
  '#cache' => [  
    'keys' => ['build_1'],  
    'contexts' => ['user'],  
  ],  
];  
  
$build_2 = [  
  '#markup' => $markup,  
  '#cache' => [  
    'keys' => ['build_2'],  
    'tags' => ['node:4'],  
  ],  
];  
  
$build_3 = [  
  '#markup' => $markup,  
  '#cache' => [  
    'keys' => ['build_3'],  
    'max-age' => '100',  
  ],  
];
```

# Cache HTML avec *Varnish*

- ***Varnish*** est un reverse-proxy qui s'occupe de servir les pages HTML mises en cache. Il peut être complexe à configurer. Il fait la même chose que le module ***Internal Page Cache***.
- ***Varnish*** est particulièrement puissant car il sert les pages plus rapidement qu'***Apache*** (qui n'est pas sollicité).
- Attention par défaut ***Varnish*** met en cache pour les anonymes seulement.
- Utiliser le module ***BigPipe*** (streaming) ou l'AJAX (requêtes serveur) pour dynamiser.
- On peut également utiliser ***Varnish ESI***.
- Avec le module ***Varnish Mobile***, on a le possibilité d'enregistrer plusieurs version d'une page (page mobile par exemple).

# Purger les caches

- Le plus compliqué n'est pas de mettre en cache, mais de purger les caches au bon moment.
- Il faut donc bien paramétrer l'affichage des modules custom (*render array*) et toujours se poser la question de la pertinence de la mise en cache.
- Les modules **Purge** et **Varnish Purger** permettent de transmettre les informations de mise en cache au reverse proxy, et donc d'invalider le cache en fonction des critères de **Drupal**.
- Attention aux dépendances qui peuvent conduire à trop purger. Utiliser un temps minimum de mise en cache peut être suffisant.



# Les modules



# Les modules

- Certains modules sont particulièrement lourds, et un seul module peut ralentir tout le système (ex. : requête complexe ou boucles imbriquées).
- Il faut donc réduire le nombre de module au maximum (dans la mesure du possible!).
- Ne pas utiliser de modules lourds pour une tâche à priori simple (module **Rules** par exemple).
- Seul le fichier **.module** est systématiquement chargé. Il contient à priori toutes les fonctions de *hook*. Le code qu'il contient représente souvent une proportion très faible du module. On ne peut donc pas dire quel est l'impact d'un module sur les performances.
- Dés-installez en production les modules contrib "UI", les modules de développement (**Devel**, **Kint**, **Web Profiler**...) et ceux qui ne servent pas!

# Performance côté front (HTML)

# Optimisation CSS

- Afin d'éviter de charger du code CSS inutile sur toutes les pages, il est conseillé de multiplier le nombre de feuilles de styles et de ne les charger que lorsque cela est nécessaire (approche **Web Component**).
- Il est plus efficace de charger une feuille de style uniquement sur l'une des pages du site en plus des fichiers de CSS communs à toutes les pages, plutôt que de l'inclure sur toutes les pages.
- La technique des Sprites CSS et l'utilisation de fonts sont conseillées également.

# Gérer l'agrégation des assets

- Il est possible d'activer l'agrégation des assets (CSS et JS) : plutôt que d'avoir une centaine (voir plus) de fichiers statiques à charger, **Drupal** va générer des agrégats.
- Potentiellement, en fonction des librairies à charger, on a plusieurs agrégats pour chaque page. Ainsi le navigateur va devoir charger plusieurs fichiers par page, sans pouvoir les mettre en cache.
- On peut agréger toutes les librairies communes à l'ensemble du site et charger ensuite celles qui sont spécifiques à une page. Ainsi le navigateur n'a plus qu'à demander ces dernières et met en cache ce qui est commun.
- La compression des CSS et des JS peut être améliorée avec le module **Advanced CSS / JS Aggregation**.

## Sans agrégation

.giore hendrerit iaculis iudus. Adau  
quadrum vindico. Aliquip exputo  
go iusto meus nunc pala pneum.

importun	CSS	128
	JS	74

2 → 65 📄 382 / 41 📄 202

## Avec agrégation

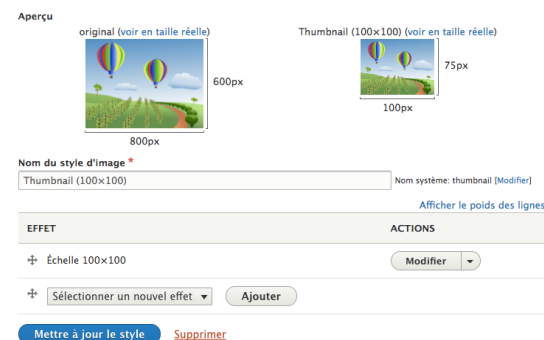
J uoioie hendrerit iaculis iudus. Adau  
za quadrum vindico. Aliquip exputo  
eligo iusto meus nunc pala pneum.

ne importunus	CSS	5
	JS	5

📄 1 → 65 📄 319 / 230 📄 10

# Gérer les images

- Il est indispensable de toujours redimensionner les images que l'on affiche. On crée pour se faire autant de styles d'image que nécessaire.
- Le module **Responsive Image** (core non activé par défaut) permet lui en plus de créer des styles d'image adaptatifs. Pour un même champ image on associe plusieurs styles d'image en fonction des points de rupture (*Breakpoints*).
- La qualité de la compression JPEG est paramétrable sur *Admin > Configuration > Média > Boîte à outils image* (75% par défaut).
- Le module **Image Style Quality**. permet de définir la qualité des images pour chaque style.



# Javascript et AJAX

- Un appel **AJAX** signifie un bootstrap complet de **Drupal**. Mais un callback AJAX est bien plus léger que la génération d'une page (aucune surcouche de thème par exemple).
- L'AJAX et la performance ou le moyen de travailler la performance serveur... ou pas ! Une ergonomie AJAX peut impliquer de nombreux appels très rapprochés. Vérifier bien la mise en cache des réponses AJAX.
- Les requêtes asynchrones en AJAX peuvent provoquer un ralentissement.
- On peut aussi utiliser le javascript pour éviter l'AJAX (validation côté client). Quand le navigateur travaille le serveur n'est pas sollicité.
- Quelques exemples de module utilisant l'AJAX : ***Quick Edit***, pagination de ***Views***, ***AJAX Comments***, ***Contact AJAX***, ***Refreshless...***



# Mesurer les performances d'un site

# Mesurer la performance Drupal avec *Web Profiler*

Le module **Web Profiler** donnent de nombreuses indications sur les ressources utilisées par un site **Drupal** :

- Temps de génération total (DNS, TCP handshake, TTFB, temps de téléchargement et temps de construction de la page).
- Temps global de requêtage et requêtes lentes.
- Emprunte mémoire.
- Performance des vues.
- Blocs et formulaires chargés.
- Configurations chargées.
- Services, événements chargés et lancés.
- Cache hit vs. Cache miss.
- Assets (css et js).
- Autre : appels externes, courriels, traductions...

PHP Config	Cache
PHP 7.1.0	
Request	
200 OK	
Timeline	
Duration: 0 ms	
Performance	
Timing	
TTFB: 7 ms	
Database	
Executed queries: 221	
User	
admin	
Views	
Total: 1	
Blocks	
Loaded: 15, rendered: 15	
Forms	

ID	hit	miss
bin.config		
webprofiler.config	1	0
language.en.webprofiler.config	1	0
language.entity.en	1	0
language.entity.fr	1	0
language.entity.und	1	0
language.entity.zxx	1	0
language.en.language.entity.en	1	0
language.en.language.entity.fr	1	0
language.en.language.entity.und	1	0

20 ms



296

in 80.15 ms



admin



0



19



0



75



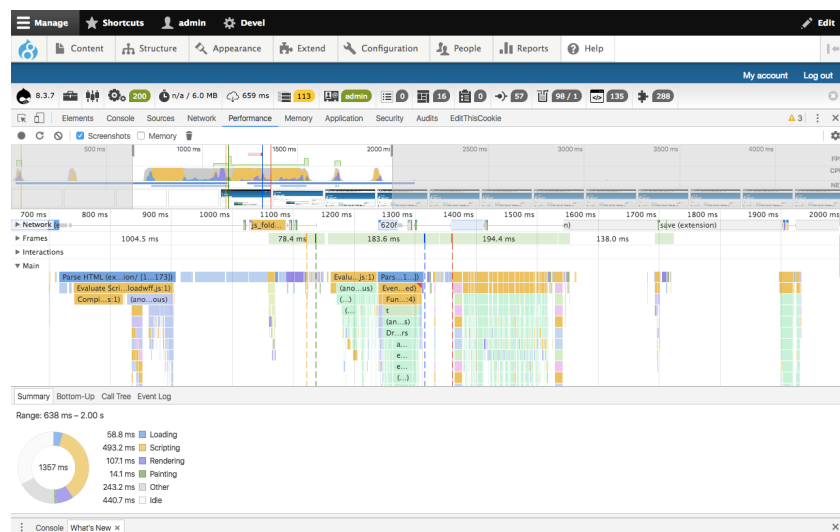
552





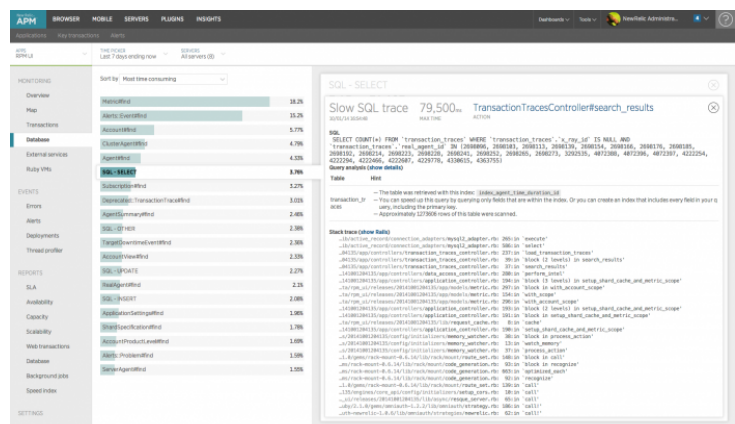
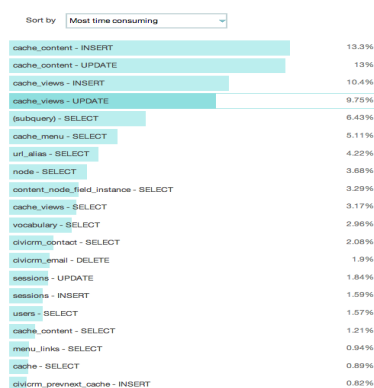
# Mesurer la performance navigateur

- Utiliser les outils de rapport de chargement navigateur ou externes.
- Attention à la rapidité de votre connexion : tout le monde n'a pas la fibre optique...
- Attention : un test local est très différent d'un test serveur (notamment pour les fichiers).
- Comparez avec d'autres sites.



# Outils externes

**New Relic**, **Blaze meter** ou **Blackfire** sont des outils payants très avancés pour aller plus loin dans l'analyse.



D'autres outils existent comme **XHprof** (gratuit, moins avancé et moins clair), monitoring-transactionnel, webwait, etc.

# Calcul en PHP

- Utiliser *microtime* pour connaître le temps d'exécution (lié au CPU dans une certaine mesure).
- Utiliser *memory\_get\_usage* pour calculer l'emprunte mémoire d'un script particulier :
  - Lancer son script plusieurs fois et moyenner.
  - Attention : le temps dépend aussi du serveur.

# Le cache ailleurs qu'en base de données

- Il est possible d'utiliser des systèmes externes pour le stockage des données de cache. Les tables en base sont ainsi déplacées en mémoire par exemple. C'est le principe de **Memcache** et **Redis**.
- Il est nécessaire d'installer les modules correspondants et de monitorer les résultats afin de bien régler ce type d'optimisation (en particulier la taille du cache et le système d'invalidation).
- Attention aux spécificités/configurations de ces systèmes. Par exemple **Memcache** ne peut pas prendre en compte les bins de plus de 1Mo. Penser à utiliser **Web Profiler** pour vérifier les cache *hits* et les cache *misses*.

# Optimiser Views

- Activer le cache de **Views**. Bien choisir ses options : tags ou max-age. Ainsi suivant le trafic sur le site, 5 minutes valent mieux que pas de cache du tout.
- **Note** : le cache de **Views** prend en compte les arguments, filtres, rôles... (contextualisation donc beaucoup plus d'entrées en cache possiblement)
- Le module **Views Custom Cache Tags** permet de prendre en compte des tags plus précis.
- Faire attention au choix de l'affichage de chaque élément : view mode des entité ou sélection des champs.
- Il est possible d'afficher le temps de requêtage d'une vue (*Admin > Structure > Views > Settings*).
- Il faut se méfier de **Views** et l'AJAX.
- La pagination de **Views** et **Views Infinite scroll** doivent être surveillés.

# Code & performance

- Toujours privilégier l'utilisation de l'injection de dépendance.
- Pour des tâches lourdes et/ou répétitives il est préférable d'utiliser *Batch API* et *Queue API*, ainsi que le *cron*.
- Attention aux boucles et aux fonctions utilisées à l'intérieur. Appeler une fonction n'est jamais neutre.
- Mettre en cache tous les affichages custom peut générer énormément de variations stockées en base. Cela est souvent contre-productif !



# Optimiser le serveur

# Le serveur

- Le mieux pour aller vite, c'est de ne rien demander à **Drupal** !
- Utiliser si possible **PHP 7**. C'est un vrai gain en performance !
- Le serveur y est pour beaucoup dans le temps de génération des pages.
- Il est possible d'installer **APC** ou équivalent.
- Utilisation de fastCGI.
- **NGINX** est une bonne solution (reverse proxy et micro-caching).
- Désactivation des modules **Apache** inutiles (si possible).
- Configurer/optimiser la base de données (nombre de requêtes simultanées...).
- Moteur de base de données pour un objectif de performance (MyISAM vs InnoDB).
- **MariaDB** est supposé être plus rapide que **Mysql**.





# Optimiser le coeur de Drupal

# Optimiser le coeur

- De nombreux modules du coeur installés par défaut peuvent être dés-installés en production :
  - *Activity Tracker*
  - *Color* (si non utilisé pour le thème)
  - *Contextual Links*
  - *Database Logging*
  - *Field UI, Views UI* (tous les modules UI non nécessaires en production)
  - Help
  - *Statistics*
  - *Syslog*
  - *Tour*

# Autres optimisations

- Optimiser le *cron* avec un module **Drupal** (*Ultimate Cron* ou autre) et s'assurer de son fonctionnement (vidage cache fichiers et BDD).
- Les modules d'optimisation du core (ex. *Taxonomy Edge* ou patches) .
- Les pages et fichiers non trouvés coûtent cher.
- Les fonctionnalités AJAX apportent d'autant plus de gains que l'installation est lourde (ne pas tout ajaxifier cependant!).
- Utilisation d'**Apache SOLR** pour la recherche (avec *Search API*), ou autre moteur d'indexation (*ElasticSearch*).
- Utilisation d'un CDN (exemple d'**Amazon**) et d'un cache front externe (ex. *cloudflare*)

# Exercices

- Tests de sites avec *www.monitoring-transactionnel.com* et **FireFox** (*Firebug*) ou **Chrome** (*PageSpeed*).
- Utilisation du module **Web Profiler** pour tester la rapidité côté serveur.
- Compression des javascripts et CSS puis des caches et vérification de l'impact.
- Activer **BigPipe**. Créer une vue des 5 derniers articles au hasard. Activer **BigPipe Demo** et vérifier l'impact en authentifié et anonyme.



# Drupal et la sécurité

# Les enjeux de la sécurité

- Qu'entendons par sécurité web ? Internet étant par définition ouvert à tous (il suffit de connaître une adresse), il est indispensable de contrôler l'accès en fonction de l'utilisateur.
- Mise en cause de la confidentialité des données.
- Mise en cause de l'intégrité des données (ajout / modification / suppression).
- Mise en cause de l'intégrité du système (altération du fonctionnement de l'installation dont l'indisponibilité et la suppression).

# Introduction

- ***Drupal*** est-il bien sécurisé ?
- ***Drupal*** est open-source, ce qui implique que son code est accessible de tous, y compris les pirates/hackers et tous les experts en sécurité.
- Il existe une équipe en charge de vérifier la sécurité du coeur de ***Drupal*** et de la plupart des modules contribs (indiqué clairement sur la page du projet en question). Pour en savoir plus : <https://www.drupal.org/drupal-security-team/general-information>.

# Les failles courantes

Les failles les plus courantes du code sont les suivantes :

- **XSS** (injection de code) : vol d'informations, redirection, bug, déclenchement d'une action.
- **Injection SQL** : modification non désirée de la base de données.
- **CSRF** (cross-site request forgery) : utilisation d'un compte à son insu afin d'effectuer des action non désirées.
- **Fixation de session** : déterminer l'identifiant de session d'un utilisateur et se connecter à sa place.



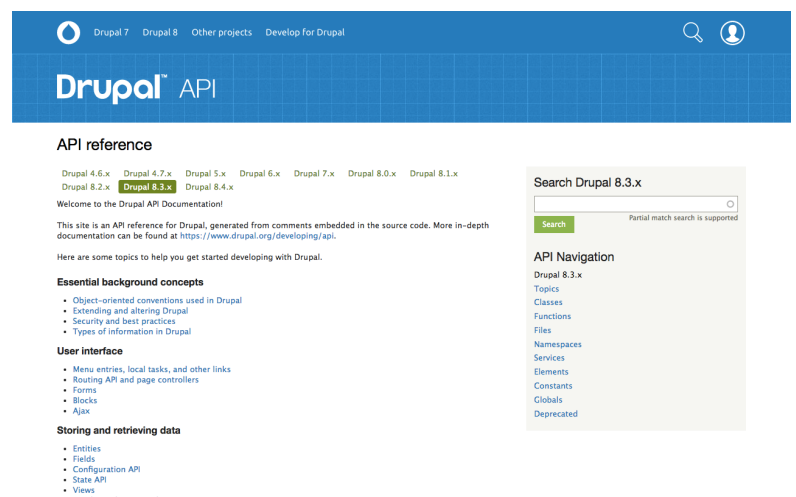
# Les réponses courantes

## Les réponses courantes :

- S'assurer que chaque URL n'est accessible (en direct) que par les « bonnes » personnes, y compris les callbacks (ajax, auto-complétions, web services...).
- Vérifier, valider et filtrer la totalité des données utilisateurs (cookies, paramètres GET / POST, données des formulaires, user-agent) en utilisation et en restitution.
- Backups réguliers du code source, de la configurations, des fichiers statiques et de la base de données sur plusieurs serveurs physiques. L'utilisation de ***GIT*** peut faciliter le travail (pour le code custom).

# Règle générale

- Toujours utiliser les APIs de **Drupal**. C'est souvent plus simple et cela évite d'introduire des failles via du code maison mal sécurisé.
- La référence à toutes les classes/fonctions est disponible sur *api.drupal.org*.





# Sécurité du code

# Coeur de Drupal

- La maintenance des versions majeures de **Drupal** est prévisible. Au minimum, la version N-1 est maintenue. Donc la version 7 sera suivie jusqu'à la sortie de **Drupal 9** (non encore annoncée).
- Security Team
- Les mises à jour de sécurité du coeur de **Drupal** sortent le 3ème mercredi du mois (s'il y a lieu!).
- Il est possible de s'abonner à la newsletter dédiée (nécessite un compte sur *drupal.org*) ou bien de suivre le compte **Twitter** @drupalsecurity.

# Les modules contrib et la sécurité

- Quelle fiabilité pour un module contrib ?
  - Les indicateurs : auteur, nombre d'utilisation, historique, activité, nombre de bugs reportés, nombre de version, date de la dernière version.
  - Attention au nombre de modules installés.
  - Attention à chaque module et à ses caractéristiques (ex. : droits d'accès du module **Views**).
- Attention aux modules en version de développement. Ils n'offrent sans doute que peu de garantie en terme de sécurité.

## Project Information

Maintenance status: [Actively maintained](#)

Development status: [Under active development](#)


Module categories: [Administration](#)

Reported installs: **45,893** sites currently report using this module. [View usage statistics.](#)


Downloads: 400,578

Automated tests: [Enabled](#)

Last modified: 2 May 2017

 Stable releases for this project are covered by the [security advisory policy](#). Look for the shield icon below.

## Downloads

Version	Download	Date
8.x-1.19 	<a href="#">tar.gz (17.07 KB)</a>   <a href="#">zip (27.02 KB)</a>	2017-Apr-06
<strong>Development releases</strong>		
8.x-1.x-dev	<a href="#">tar.gz (19.5 KB)</a>   <a href="#">zip (29.81 KB)</a>	2017-Jun-20

[View all releases](#)

# Accès à la base de données

- Il faut absolument toujours passer par la couche d'abstraction de la base de données. Éviter l'utilisation de la fonction *db\_query()*. On ne doit jamais faire de requêtes directement en base.
- Lorsque l'on souhaite manipuler un type d'entité (noeud, utilisateur, bloc, vue...) il est nécessaire d'utiliser **Entity API** (service *entity\_type.manager*) plutôt que de faire manuellement des requêtes en base de données. C'est plus simple et cela garantit un certain nombre de contrôles automatiques.

```
public function query($group_by = FALSE) {
    $this->ensureMyTable();

    // Use the table definition to correctly add this user ID condition.
    if ($this->table != 'comment_field_data') {
        $subselect = $this->database->select('comment_field_data', 'c');
        $subselect->addField('c', 'cid');
        $subselect->condition('c.uid', $this->argument);

        $entity_id = $this->definition['entity_id'];
        $entity_type = $this->definition['entity_type'];
        $subselect->where("c.entity_id = $this->tableAlias.$entity_id");
        $subselect->condition('c.entity_type', $entity_type);

        $condition = (new Condition('OR'))
            ->condition("$this->tableAlias.uid", $this->argument, '=')
            ->exists($subselect);

        $this->query->addWhere(0, $condition);
    }
}
```

```
/* The name of a theme.
 */
function block_theme_initialize($theme) {
    // Initialize theme's blocks if none already registered.
    $has_blocks = \Drupal::entityTypeManager()->getStorage('block')->loadByProperties(['theme' => $theme]);
    if (!$has_blocks) {
        $default_theme = \Drupal::config('system.theme')->get('default');
        // Apply only to new theme's visible regions.
        $regions = system_region_list($theme, REGION_VISIBLE);
        $default_theme_blocks = \Drupal::entityTypeManager()->getStorage('block')->loadByProperties(['theme' => $default_theme]);
        foreach ($default_theme_blocks as $default_theme_block_id => $default_theme_block) {
            if (strpos($default_theme_block_id, $default_theme . '.') == 0) {
                $id = str_replace($default_theme, $theme, $default_theme_block_id);
            }
            else {
                $id = $theme . '.' . $default_theme_block_id;
            }
            $block = $default_theme_block->createDuplicateBlock($id, $theme);
            // If the region isn't supported by the theme, assign the block to the
            // theme's default region.
            if (!isset($regions[$block->getRegion()])) {
                $block->setRegion(system_default_region($theme));
            }
            $block->save();
        }
    }
}
```



# Sécurité du serveur

# Sécurité serveur

- Le serveur doit être régulièrement mis à jour, comme tout logiciel.
- Les fichiers doivent être à “read only” pour **Apache**, et les droits d’écriture doivent être possédés par un utilisateur séparé. Cf. <http://drupal.org/node/244924>.
- Les informations sensibles du fichier de settings peuvent être contenu dans un fichier différent hors de l’arborescence web.
- Le dossier de synchronisation de la configuration (**/sites/default/files/config\_XXX/sync**) devrait également être placé en dehors de l’arborescence web.
- Il est nécessaire de supprimer physiquement les modules non utilisés (cf. le module **Coder**).
- On ne devrait **plus jamais** utiliser de client FTP.

```
*  
* Example:  
* @code  
*   $settings['hash_salt'] = file_get_contents('/home/example/salt.txt');  
* @endcode  
*/  
$settings['hash_salt'] = '_PHo4cY0p49Wq2rKmrJfi3nd18R81WCsh7D34xPeepjv6jC5Wg-4L0KqTzTZpSWg2nDIefk0UA';
```



# Les fichiers privés

- Lorsque l'on utilise les fichiers privés, il est nécessaire de :
  - définir dans le fichier `settings.php` le répertoire correspondant. Il est préférable de le localiser hors de l'arborescence web afin qu'il ne soit pas accessible depuis une URL.
  - vérifier la conformité du fichier `.htaccess`. Attention en cas d'utilisation de **NGINX**.
- Il faut bien faire attention à restreindre les extensions des fichiers chargés (pas d'exécutables et se méfier des fichiers PDF).
- L'utilisation des fichiers privés a un impact sur les performances du système. Il ne faut donc les utiliser que lorsque cela est nécessaire.

```
*  
* See https://www.drupal.org/documentation/modules/file for more information  
* about securing private files.  
*/  
$settings['file_private_path'] = '../mon-dossier-prive';
```



# Gestion des droit d'accès

# Les droits d'accès

- Chaque module arrive avec ses propres droits d'accès.
- Droit d'accès de base : « Accéder au contenu publié ». Si non offert : création d'un intranet.
- Aucun droit d'accès n'est offert par défaut (seuls les utilisateurs ayant le rôle **Administrator** ont tous les droits). Attention un module peut dépendre d'un autre droit d'accès (*administrer site configuration*).
- Certains droits d'accès peuvent en englober d'autres (ex. : permission "*administrer les commentaires*" bypassse toutes les autres restrictions, idem pour "*administrer les contenus*", "*administrer les utilisateurs*", etc.).
- De nombreux modules existent pour gérer des droits d'accès plus finement.
- Dans un module *custom* il est possible d'afficher un texte spécifique sur les droits d'accès touchant plus profondément à la sécurité.
- **Attention** : toutes les permissions du rôle authentifié sont données à tous les rôles.

# Criticité des droits d'accès

- Les droits d'accès critiques permettent de prendre le contrôle du site :
  - Les accès à l'administration des modules.
  - Accès à l'administration des utilisateurs et / ou des permissions.
  - Administrer la configuration du site : utilisé par défaut par beaucoup de modules.
  - Administrer les filtres.
- Les droits d'accès semi-critiques peuvent permettre de "casser" le site :
  - Accès aux blocs.
  - Accès aux type de contenus, views, taxonomie, etc...
- Les droits d'accès incompris :
  - Exemples : accès en administration des utilisateurs / des taxonomies => accès à la gestion des champs. Administration des menus permet aussi de supprimer des menus
  - Accès à l'administration des blocs => accès à l'administration de tous les blocs sans exceptions.
  - Accès à l'administration des menus => suppression possible.

# Méthodes *access()*

- Les entités de configuration comme de contenus disposent d'une méthode *access()* (ou équivalent par exemple *blockAccess()*).
- Celle-ci peut être étendue.
- Il est donc simple de vérifier/contrôler programmatiquement l'accès.
- Il faut implémenter cette méthode dès que nécessaire. Ex. utilisation pour les blocs.

# Méthodes *blockAccess()*

- On renvoi simplement un objet *AccessResult* avec l'une de ses méthode statique : *allowed()*, *forbidden()*, *neutral()*... (<https://api.drupal.org/api/drupal/core!lib!Drupal!Core!Access!AccessResult.php/class/AccessResult/8.x.x>).
- L'altération est également possible en utilisant les fonctions *hook\_entity\_access()*, *hook\_node\_access()*, *hook\_block\_access()*, *hook\_entity\_create\_access()*, etc...

```
/**
 * {@inheritdoc}
 */
protected function blockAccess(AccountInterface $account) {
    $route_name = $this->routeMatch->getRouteName();
    if ($account->isAnonymous() && !in_array($route_name, ['user.login', 'user.logout'])) {
        return AccessResult::allowed()
            ->addCacheContexts(['route.name', 'user.roles:anonymous']);
    }
    return AccessResult::forbidden();
}
```

# Utiliser des modules

- De nombreux modules existent pour restreindre les accès et/ou ajouter davantage de permissions :
  - *Content Access.*
  - *Field Permissions.*
- Bien les tester dans tous les scénarios possibles (créer autant d'utilisateurs que nécessaires ayant chacun des rôles différents).
- Ces tests peuvent être très longs. Ils n'en demeurent pas moins indispensables.

# Routing

- Tout chemin d'accès doit être contrôlé.
- Il faut utiliser systématiquement la propriété « *\_requirements* ».
- Le contrôle d'accès est souvent fait grâce au système de rôles/permissions. On peut également ajouter en option un token de contrôle d'accès.

```
user.logout.http:  
  path: '/user/logout'  
  defaults:  
    _controller: \Drupal\user\Controller\UserAuthenticationController::logout  
  methods: [POST]  
  requirements:  
    _user_is_logged_in: 'TRUE'  
    _format: 'json'  
    _csrf_token: 'TRUE'
```





# Injection de données utilisateur

# Les formulaires

- La première porte d'entrée d'un site est les formulaires (connexion, commentaires, création de contenu...). Il est donc nécessaire de bien les sécuriser.
- La **Form API** permet de s'assurer que les bonnes pratiques sont respectées (token de validation comportant une clé privée + ID de session + string associé à l'action), même si cela n'empêche pas de « blinder » au niveau de la validation des données.

```
/**
 * {@inheritdoc}
 */
public function buildForm(array $form, FormStateInterface $form_state) {
    $config = $this->config('system.site');

    // Display login form:
    $form['name'] = [
        '#type' => 'textfield',
        '#title' => $this->t('Username'),
        '#size' => 60,
        '#maxlength' => USERNAME_MAX_LENGTH,
        '#description' => $this->t('Enter your @s username.', ['@s' => $config->get('name')]),
        '#required' => TRUE,
        '#attributes' => [
            'autocorrect' => 'none',
            'autocapitalize' => 'none',
            'spellcheck' => 'false',
            'autofocus' => 'autofocus',
        ],
    ],
];

    $form['pass'] = [
        '#type' => 'password',
        '#title' => $this->t('Password'),
        '#size' => 60,
        '#description' => $this->t('Enter the password that accompanies your username.'),
        '#required' => TRUE,
    ],
];

    $form['actions'] = ['#type' => 'actions'];
    $form['actions']['submit'] = ['#type' => 'submit', '#value' => $this->t('Log in')];

    $form['#validate'][] = '::validateName';
    $form['#validate'][] = '::validateAuthentication';
    $form['#validate'][] = '::validateFinal';

    $this->renderer->addCacheableDependency($form, $config);

    return $form;
}
```

# Les filtres Drupal

- Principe des filtres dans **Drupal** : **le filtrage se fait a posteriori.**
- Les filtres par défaut :
  - *Plain Text* : rien ne passe.
  - *Restricted HTML* : stricte minimum (pas d'images par ex.).
  - *Basic HTML* : tous les essentiels de la contribution en HTML (y compris images mais limité au site courant).
  - *Full HTML* : Attention, tout passe !
- Possibilité de créer autant de filtre que souhaité. IL est nécessaire de bien régler les interaction avec **CKeditor**.
- Les filtres sont assignés à des rôles et s'appliquent à tous les champs filtrés. Ils servent également à d'autres usages que la sécurité.
- A noter : si un utilisateur a accès à l'édition d'une entité, mais pas au filtre choisi sur un champ texte filtré, il ne verra pas le champ.

# XSS et API

Afin d'éviter les failles de type XSS, vous devez utiliser les fonctions de l'API dans le code :

- fonctions `t()` et `\Drupal::translation()->formatPlural()` avec les placeholders `@` (texte brut) ou `%` (texte brut avec la balise `<em>`). Attention au placeholder `!` qui ne filtre rien.
- fonction `Html::escape()` pour du texte brut.
- fonction `Xss::filter()` pour du texte acceptant le HTML.
- fonction `Xss::filterAdmin()` pour du texte acceptant l'écrasante majorité des tags HTML (filtre les balise de scripts par exemple).

```
// New administrative account without notification.
if ($admin && !$notify) {
  drupal_set_message($this->t('Created a new user account for <a href=":url">%name</a>. No email has been sent.',
    [':url' => $account->url(), '%name' => $account->getUsername()]));
}
// No email verification required: log in user immediately.
```

# Utiliser les données filtrées

- Il faut coder selon les standards **Drupal**.
- Il faut toujours utiliser les fonctions de l'API **Drupal**.
- Le module **Coder** (<https://www.drupal.org/project/coder>) permet de s'assurer que ces standards sont bien respectés.

- @file block missing ([Drupal Docs](#))
- Line 28: Control statements should have one space between the  

```
switch($section) {
```
- Line 81: string concatenation should be formatted with a space :  

```
'#default_value' => variable_get('simpletestauto_s
```
- Line 105: missing space after comma  

```
if (stristr($msg,'Error')) {
```
- Line 106: Use an indent of 2 spaces, with no tabs  

```
if (stristr($msg,'password')) {
```
- Line 106: missing space after comma  

```
if (stristr($msg,'password')) {
```

# Exercice

- Insérer une balise `<script>` dans un contenu
  - Insérez dans une page avec le filtre FULL HTML le code suivant :  
`<script>alert('Hack !');</script>`
  - Affichez la page.
  - Allez dans la configuration du filtre pour limiter les balises HTML autorisées.
  - Réaffichez la page.
- Mise en place des fichiers privés :
  - Déclarer le chemin de fichier privé dans le settings.php.
  - Ajouter un champ fichier joint privé au type de contenu page.
  - Créer une page et attachez un fichier : tester d'y accéder en mode anonyme quand la page est publiée / dépubliée..



# Gestion des connexions

# Les mots de passe

- Les mots de passe sont encryptés et indécryptables (utilisation d'un log2 à 16 ; ce service peut être personnalisé).
  - Module **Password Policy** pour les mots de passe utilisateur.
  - Pas d'utilisation de pseudo type admin / superadmin / nom du site...
- Faire attention aux mots de passe des administrateurs. Ne pas disposer d'un trop grand nombre de comptes administrateur.
- Pas de possibilité de deviner les pseudos / adresse e-mail des utilisateurs administrateurs.
- Ne pas utiliser le même mot de passe pour plusieurs applications.
- Utilisation du protocole HTTPS et du module **Secure Login**.
- Réduire la durée de session (</sites/default/settings.php>) : 1 ou 2 jours, mais pas plus.
- La table **FLOOD** permet de contrer les attaques par « force brute ».



# Les sessions utilisateur

- **Drupal** gère les sessions utilisateur de manière sécurisée (SHA-256, base64encoded, URL-friendly). Il faut limiter leur manipulation. **Drupal** génère un ID de session enregistré en base de données (table *sessions*) et par le navigateur.
- Une connexion se fait donc entre un site et un navigateur
- Si l'on souhaite manipuler des sessions, on dispose de la classe *SessionManager*.
- Il ne faut **jamais afficher les ID de session** : une session peut être définie à loisir côté navigateur.
- Les sessions sont cryptées en base de données.

# Les permissions

- Permission simple à déclarer dans le fichier *mon\_module.permissions.yml* :

```
bypass node access:  
  title: 'Bypass content access control'  
  description: 'View, edit and delete all content regardless of permission restrictions.'  
  restrict access: true
```

- *Restrict access* est à utiliser pour les permissions critiques.
- Possibilité de déclarer un fichier de permissions définies dynamiquement en PHP.

```
permission_callbacks:  
  - \Drupal\node\NodePermissions::nodeTypePermissions
```

- Dans le code on peut tester les permissions d'un utilisateur de la façon suivante :

```
$account->hasPermission('administer nodes')
```

# Permissions dynamiques

```

* The node type permissions.
* @see \Drupal\user\PermissionHandlerInterface::getPermissions()
*/
public function nodeTypePermissions() {
    $perms = [];
    // Generate node permissions for all node types.
    foreach (NodeType::loadMultiple() as $type) {
        $perms += $this->buildPermissions($type);
    }

    return $perms;
}

/**
 * Returns a list of node permissions for a given node type.
 *
 * @param \Drupal\node\Entity\NodeType $type
 *   The node type.
 *
 * @return array
 *   An associative array of permission names and descriptions.
 */
protected function buildPermissions(NodeType $type) {
    $type_id = $type->id();
    $type_params = ['%type_name' => $type->label()];

    return [
        "create $type_id content" => [
            'title' => $this->t('%type_name: Create new content', $type_params),
        ],
        "edit own $type_id content" => [
            'title' => $this->t('%type_name: Edit own content', $type_params),
        ],
        "edit any $type_id content" => [
            'title' => $this->t('%type_name: Edit any content', $type_params),
        ],
        "delete own $type_id content" => [
            'title' => $this->t('%type_name: Delete own content', $type_params),
        ],
        "delete any $type_id content" => [
            'title' => $this->t('%type_name: Delete any content', $type_params),
        ],
    ];
}
```



# Sécurité côté front avec *TWIG*

# XSS et TWIG

- **TWIG** filtre automatiquement : « auto-escaping ». Sa simple utilisation sécurise le site.
- Tout code HTML doit être affiché via un fichier **TWIG** (éviter le HTML dans *#markup*, utiliser à la place *#plain\_text*) ou bien avec la propriété *#template* dans un render array.
- Quelques filtres supplémentaires :
  - raw
  - t
  - clean\_class / clean\_id
  - Filter {{ item|render|filter }}



# Sécuriser son site en production

# En faire plus en production

- Il est conseillé de désactiver les modules non utiles en production (développement, UI & non utilisés). On limite ainsi l'accès à certaines pages du back-office et on soulage également le site (gain de performances).
- Afin de ne pas exposer de réponses 403 (Accès refusé), qui pourraient inciter à vouloir forcer l'accès, on peut transformer toutes ces réponses en 404 (Page non trouvée) avec le module **403 to 404** (<https://www.drupal.org/project/m4032404>).
- Utiliser les modules **Database email Encryption** et **Field encryption, Cryptolog** (cryptage des lps).
- Restreindre certains éléments du site à certaines IP (**IP Range**), comme l'administration ou pour certains rôles (**Restrict Login** ou **Role Access by IP**).
- Vérification régulière des tentatives de connexion échouées. Installation possible du module **Login History**.
- On peut utiliser le module **Rename Admin Paths** pour renommer les chemins du back-office.

# En faire encore plus...

- Renforcer la connexion avec ***Login Security*** (blocage de la connexion par IP et baisse du nombre de connexion avant de bloquer un compte).
- ***Session Limit*** permet de limiter le nombre de sessions par rôle utilisateur.
- Le module ***Configuration Read-Only Mode*** interdit la modification des fichiers de configuration.
- Bloquer définitivement certaines pages du back-office (par exemple les pages de listing et de dés-installation des modules).
- Configurez votre site pour disposer de *Secure Site* ou *Shield* (ou .htaccess) en développement.
- Installez ***Maillog*** en développement.
- Installation de HTTP Response header.



# Configuration du fichier *settings.php*

- Le fichier */sites/default/settings.php* (ou équivalent) regroupe plusieurs configurations de sécurité. **Ce fichier doit être toujours en lecture seule.**
- Une clé de sécurité spéciale pour l'installation `$settings['hash_salt']` dans le `settings.php`. `$settings['hash_salt'] = file_get_contents('/home/example/salt.txt');` possible.
- Définir les *Trusted host pattern* :

```
$settings['trusted_host_patterns'] = array(
    '^example\.com$',
    '^.+\.example\.com$',
    '^example\.org$',
);
```
- Désactiver en « dur » la mise à jour des modules (et désactiver le module ***Update Manager***) :

```
settings['allow_authorize_operations'] = FALSE;
```
- Fichiers publics et privés.
- Ne pas afficher les erreurs.

# Utilisateur superadmin

- On ne devrait jamais utiliser un compte ayant tous les droits sur le système. Si celui-ci se fait hacké, alors on n'a plus aucune protection.
- Il est donc préférable de n'activer que des utilisateurs ayant des accès restreints.
- Faire bien attention à la permission « *Administrer les utilisateurs* », car elle permet aussi d'avoir la main sur les utilisateurs administrateurs. Préférer l'utilisation d'un module comme ***Administer Users by role***.

# La recherche

- Il est nécessaire de faire attention à la recherche :
  - Indexation : données non affichées mais peut donner des informations.
  - Affichage : s'assurer du contrôle des accès.
- Recherche Google : attention au fichier **robots.txt** et à la mise en cache de Google. Attention également à l'indexation d'environnement de développement et/ou de pré-production. Utiliser le fichier **.htaccess**.

# Sécurité et recette

- Importance de la recette, seul moyen de s'assurer des droits d'accès des utilisateurs :
  - Spécification en amont des droits : document de référence obligatoire.
  - Utilisation du module *Masquerade* pour test (à désactiver en production).
  - Aller sur admin/index avec chaque rôle.
- La recette est la seule manière de vérifier les accès réels des utilisateurs tant en termes de code que de configuration.

# Trop tard...j'ai été hacké

# J'ai été hacké...

- Utilisation de **GIT** pour vérifier les fichiers ajoutés / modifiés / supprimés.
- Rechargement des librairies via **Composer** si ces dernières n'étaient pas dans **GIT**. Comparaison des librairies sur le site et la version disponible. Cf. également le module **Composer Security Checker**.
- Vérification de la présence de fichiers exécutables dans les fichiers du site.
- Visualisation des configurations en base de données vs. la dernière version des configurations.
- Modification de tous les mots de passe (**Drupal**, serveur, etc...).
- Vérification/rechargement des contenus. En particulier les contenus avec un filtre de type *Full HTML* ou équivalent.
- Il faut disposer de **GIT** et de backup propres réguliers.

# J'ai été hacké...

- Avec les modules **Hacked!** et **Diff**, on a la possibilité de comparer les modules de l'installation avec la version équivalente en ligne sur *drupal.org*.
- Eventuellement, avec **Composer**, on peut reconstruire entièrement l'installation de **Drupal**, des librairies et des modules contribués (toutes les librairies PHP compatibles en réalité).

modules/blog/blog.module	Deleted ❌
modules/blog/blog.info	Deleted ❌
modules/dblog/dblog.info	Changed! ⚠️
modules/node/node.module	Changed! ⚠️
robots.txt	Changed! ⚠️
install.php	Changed! ⚠️
.htaccess	Changed! ⚠️
index.php	Changed! ⚠️
scripts/cron-lynx.sh	Unchanged ✓
scripts/drupal.sh	Unchanged ✓

# Exercice

- Installer les modules **Hacked!** et **Diff**.
- Modifiez un de vos fichiers dans un module contrib ou du coeur de **Drupal**.
- Sur **Admin > Rapports > Hacked**, lancez **Hacked!**. Attention le processus peut être relativement long.
- Installer le module **Security Review** et lancer le test de vérification du système. Quel rapport fournit-il ? Comment corriger les erreurs détectées ?

## Themes

**Business** 8.x-1.7

0 files changed, 0 files deleted

Includes: *Business*

Unchanged ✓

[View details of changes](#)

## Uninstalled modules

**AddToAny Share Buttons** 8.x-1.8

1 file changed, 22 files deleted

Includes: *AddToAny*

Changed! ✖

[View details of changes](#)

**Administer Users by Role** 8.x-2.0-alpha4

0 files changed, 12 files deleted

Changed! ✖

[View details of changes](#)



## Trained People c'est aussi :

- des **formations Drupal spécialisées** (Webmaster, Développeur front, Développeur back, Sécurité & Performance, Déploiement & Industrialisation).
- des **certifications à Drupal** (Webmaster, Themeur/Intégrateur, Développeur et Expert).
- un programme de **e-learning Symfony/Drupal 8** en partenariat avec **SensioLabs**.
- de **l'accompagnement** durant vos projets (audit, régie...).
- des **recommandations** (freelances, agences, hébergement).



**TRAINEDPEOPLE**

**SensioLabs**  
**UNIVERSITY**

**Merci !**