Drupal 8
Déploiement

Animée par Romain JARRAUD

Romain JARRAUD

Formateur / Consultant

TRAINEDPEOPLE

Au programme…

- Identifier les différents éléments impliqués dans un déploiement.
- Comprendre le système de gestion de la configuration de **Drupal 8** et ses limitations par rapport au déploiement.
- Connaitre les modules additionnels dédiés au déploiement.
- Être capable de créer un script de déploiement.

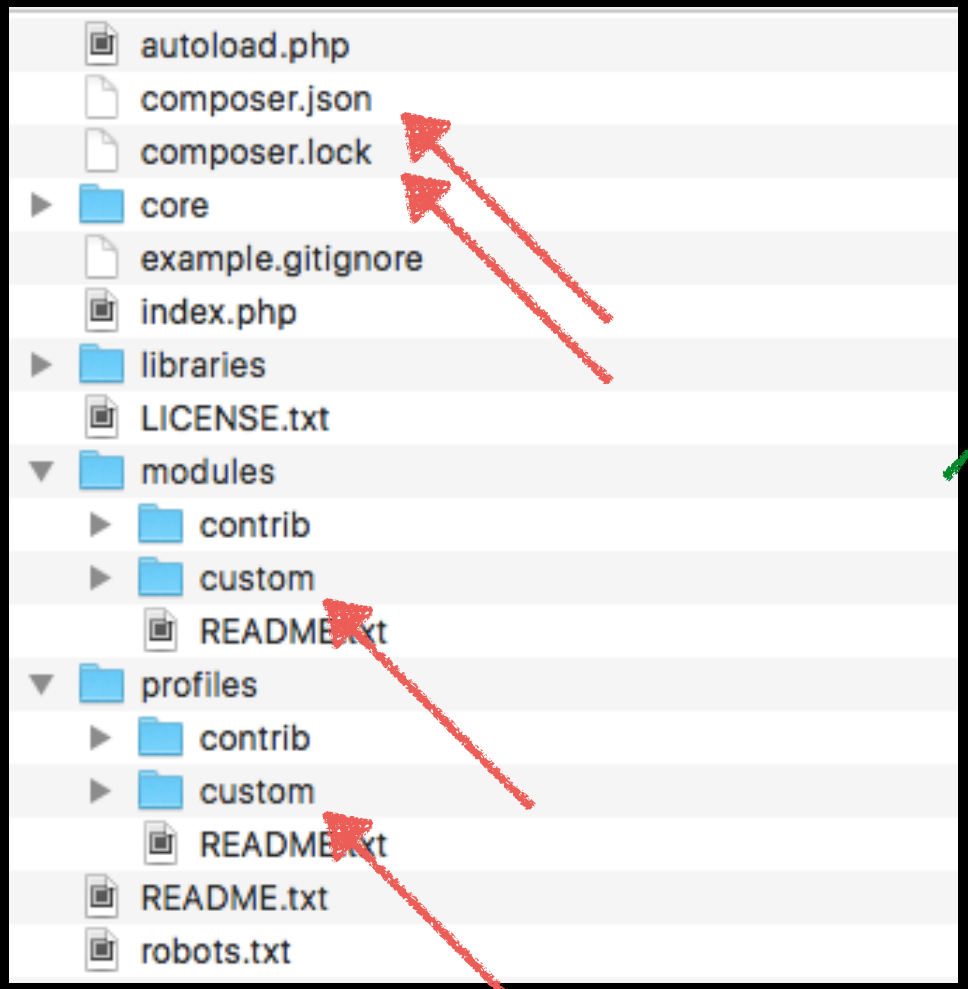
Déploiement

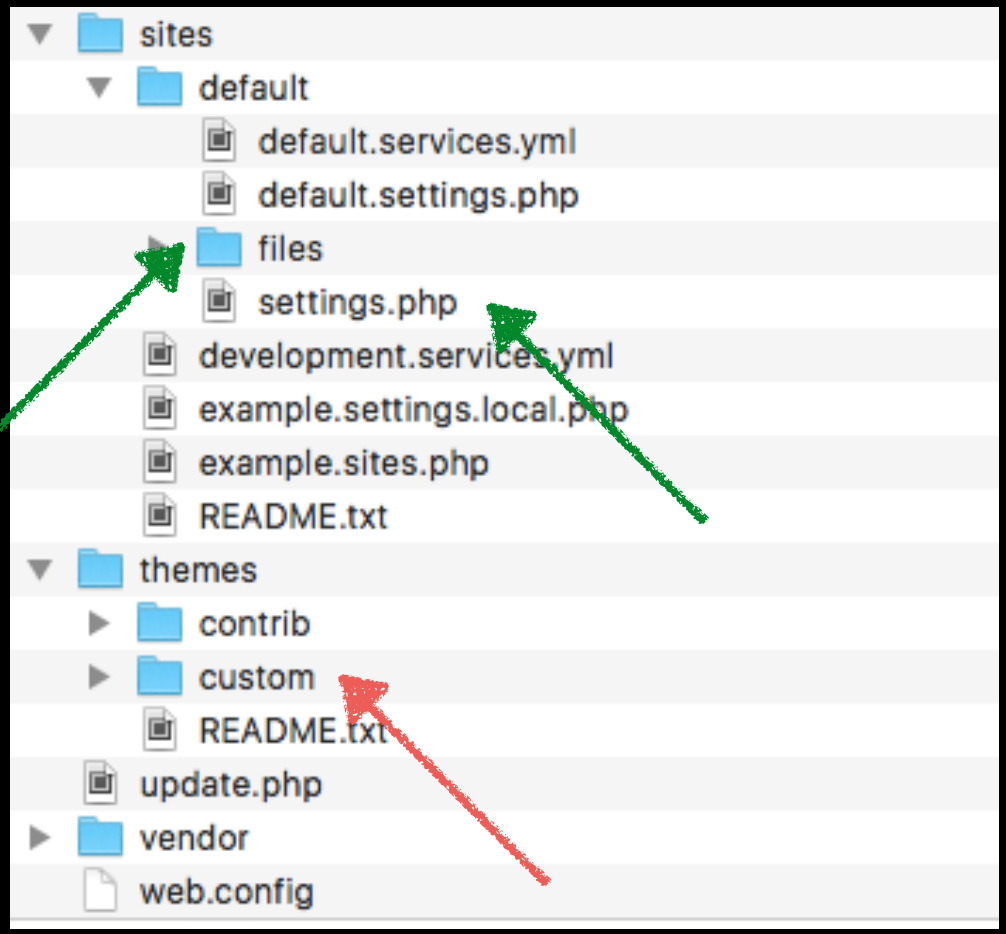
- Un site ***Drupal*** est l'association entre une base de données et des fichiers.
- Le déploiement est l'art de mettre à jour un site (staging, pré-production, production…) sans aucune perte depuis un environnement local.
- Lors d'un déploiement, de nouvelles fonctionnalités et/ou contenus doivent être intégrés sur les différents environnements.
- Il est **impossible de faire des dumps** de base de données, car on perd alors tous les changements en production (ajout de contenus, création de configuration à chaud…). De plus nous allons voir que la configuration de chaque environnement est bien particulière.
- Le **déploiement** doit donc se faire à partir de **fichiers** que l'on peut versionner.

Structure des fichiers

- On peut séparer les fichiers d'un site en plusieurs catégories :
  - coeur de Drupal.
  - fichiers de settings (/sites/default/settings.php…).
  - modules, thèmes et profiles contrib.
  - modules, thèmes et profiles custom.
  - librairies PHP externes.
  - librairies JS externes.
  - fichiers statiques de contenu (images, PDFs…).
- Seuls les fichiers de code custom et les fichiers statiques (uploadés par les utilisateurs) sont non récupérables depuis un dépôt de fichiers. Tous les autres peuvent être re-téléchargés à n'importe quel moment.

Structure des fichiers





Fichiers propres au site

Fichiers locaux

Outils pour le déploiement

- **Drush** (**Dru**pal **Sh**ell): administration d'un site ***Drupal*** en ligne de commande. A installer sur tous les environnements.
- **GIT** : versioning de fichiers. A installer sur tous les environnements.
- **Composer** : gestionnaire de librairies PHP. Pas obligatoire, mais permet de récupérer tous les fichiers d'un projets à partir d'un fichier ***/composer.json***. ***Composer*** est utile pour tous les environnements sauf celui de production (on évite de tester le code de nouveau). Dans le cas où l'on utilise ***Composer***, il ne faut plus installer de modules via l'interface ou via ***Drush***. Tout doit être géré par le fichier les ***/composer.json*** et ***/composer.lock*** qui sont donc lui versionnés.

Composer

Composer

- ***Composer*** est un gestionnaire de libraries PHP. Il permet de récupérer le code source des libraries compatibles en ligne de commande et gère toutes les éventuelles dépendances.
- Le cœur de Drupal utilise ***Composer*** pour charger les différentes librairies PHP dont il a besoin pour fonctionner.
- ***Composer*** est lui-même une librairie PHP présente dans le dossier ***/vendor***. Elle n'est pas spécifique à ***Drupal*** et est utilisable dans d'autres projets PHP.
- ***Composer*** permet aussi de gérer la mise à jour des librairies PHP nécessaires au cœur (***Symfony***, ***Guzzle***, etc…), comme le cœur lui-même et les modules contrib.
- Par défaut, toutes les librairies gérées sont présentes dans le dossier ***/vendor*** à la racine du projet.

Composer

- La liste des libraries nécessaires à un projet est définie dans le fichier */composer.json*.
- Une fois que ces librairies ont été installées, *Composer* crée un fichier */composer.lock* contenant la liste des librairies et leurs versions actuellement installées.

Configuration de *Composer*

- La configuration est définie en premier lieu dans le fichier ***/composer.json*** qui est à la racine du projet. Il ne fait pas partie du cœur de ***Drupal*** et a vocation à être modifié pour les besoins du projet.
- D'autres fichiers *composer.json*, à divers endroits, viennent le compléter:
  - dans le cœur : ***/core/composer.json***
  - dans les librairies déclarées dans le ***/composer.json*** et présentes dans ***/vendor.***

Et éventuellement, si ceux-ci sont déclarés dans le ***/composer.json***

  - dans les modules contrib et custom.
  - dans les thèmes contrib et custom.
  - dans les profiles contrib et custom.

Configuration de *Composer*

- Indiquer qu'une partie des dépendances sont à récupérer auprès d'un nouveau dépôt comme par exemple ***https://packages.drupal.org/8***, plutôt que sur le dépôt par défaut ***Packagist***.
- Comme le dossier d'installation par défaut est ***/vendor***, on peut indiquer que le cœur, les modules et les thèmes **Drupal** doivent être installés à des emplacements spécifiques.

```
},
"repositories": [
    {
        "type": "composer",
        "url": "https://packages.drupal.org/8"
    }
]
```

```
        "merge-extra": false
    },
    "installer-paths": {
        "core": ["type:drupal-core"],
        "modules/contrib/{$name}": ["type:drupal-module"],
        "profiles/contrib/{$name}": ["type:drupal-profile"],
        "themes/contrib/{$name}": ["type:drupal-theme"],
        "drush/contrib/{$name}": ["type:drupal-drush"],
        "modules/custom/{$name}": ["type:drupal-custom-module"],
        "themes/custom/{$name}": ["type:drupal-custom-theme"]
    }
}
```

Composer : l'outil central de gestion des dépendances ?

- Dans l'absolu, ***Composer*** offre une solution de gestion des librairies plus complète que ***Drush***, puisqu'il permet, en plus de gérer ***Drupal***, ses thèmes et ses modules, de gérer des dépendances à des libraires PHP tierces. Il est en outre possible de lui faire gérer des librairies JS ou des exécutables «serveur», appliquer des patches, lancer des scripts, etc...
- Le développement d'applications web consistant de plus à plus en l'intégration de divers composants open source, ***Composer*** s'impose comme l'outil central idéal.
- Il facilite l'intégration de solutions comme ***Solr***, ***Elasticsearch***, ***Selenium***, ***PhantomJS***... et leur maintenance, qui nécessitent des dépendances de natures diverses pour fonctionner.

Composer : l'outil central de gestion des dépendances?

- Certains développeurs choisissent d'utiliser ***Composer*** pour gérer le code de l'ensemble de leur projet. Dans ce cas tout doit être installé par ***Composer***.
- À noter que ***Composer*** ne fait que télécharger et mettre à jour les modules et thèmes **Drupal**. Les commandes *drush en module* et *drush updb* seront encore nécessaires pour activer les modules téléchargés et lancer les mises à jour associées en base de données.
- **Attention** : ***Composer*** et ses différentes commandes réclament d'être maîtrisés. Leur utilisation peut faire surgir de nouveaux problèmes. Son utilisation pourrait inutilement compliquer la gestion de projets simples.

Composer et déploiement

- En local :
  - Après l'ajout d'une nouvelle dépendance (***composer require***), la commande ***composer update*** est lancée dans l'environnement local. Elle réalise la re-génération d'un ***/composer.lock*** par la fusion de l'ensemble des fichiers ***composer.json*** des différentes librairies du projet, contrôlant que les différentes dépendances déclarées sont en cohérence. Les différentes librairies sont alors installées et/ou mises à jour.
  - Le fichier ***/composer.lock*** généré pourra être commité si, après tests, tout s'avère fonctionner correctement.
- En staging, pre-production, production :
  - La commande ***composer install*** installe les dépendances sur la base du fichier ***/composer.lock***.
  - Attention: la commande ***composer update*** ne doit pas être utilisée sur les instances de dev, staging, prod. Elle risquerait de créer de grosses anomalies en téléchargeant des versions plus récentes mais non testées sur le projet en question.

Quel(s) fichier(s) versionner ?

- */composer.lock*

Il peut être versionné. Drupal n'en a pas besoin pour fonctionner. Mais il est indispensable si le contenu du dossier *./vendor* n'est pas versionné afin de télécharger les librairies et qu'on a choisi d'imposer de lancer un *build* dans chacun des environnements cibles (local, intégration, pre-prod, prod, etc.).

- */composer.json*

Il n'est en soi pas utile en pré-production et production, mais on peut considérer que c'est une source que doivent pouvoir éditer les collaborateurs, et à ce titre il peut être versionné.

- */vendor*

On peut considérer qu'il alourdit inutilement le dépôt. On peut à l'inverse considérer que le versionner limite le nombre d'erreurs qui peuvent survenir et réduit la longueur du processus d'update. De plus, l'ignorer implique de s'assurer que *composer* fonctionne parfaitement sur tous les environnements impliqués (firewall, etc.).

- */modules/contrib*, */themes/contrib*

Mêmes considérations que pour le dossier */vendor*.

Composer en conclusion

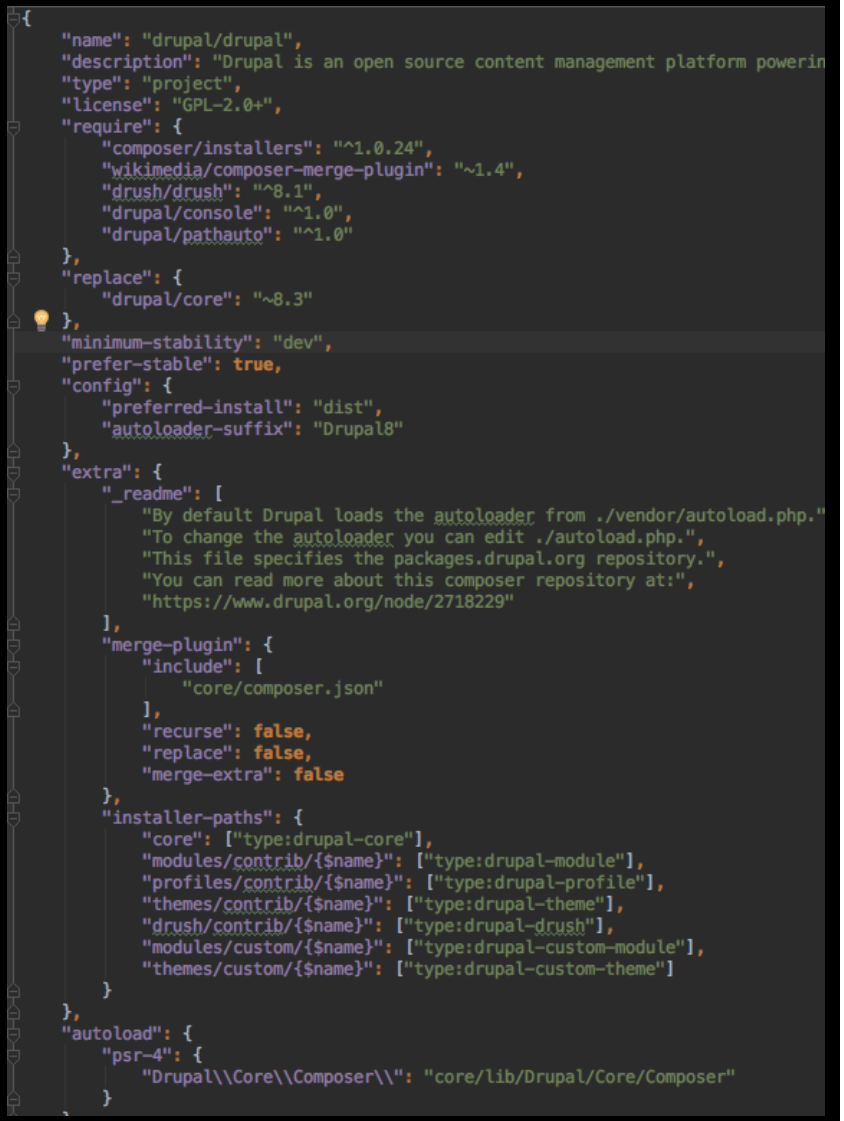
- C'est à vous de décider si vous voulez utiliser ***Composer*** et avec quelle extension.
- Sur un projet, commencez à l'utiliser pour un périmètre limité et allez-y progressivement, car l'outil demande d'être correctement maîtrisé par tous les collaborateurs !
- Pour accélérer ***Composer*** avec ***Drupal*** il est conseillé d'installer la librairie

***zaporylie/composer-drupal-optimizations*.**

Workflow avec GIT

Avec ou sans *Composer* ?

- Les fichiers que l'on versionne diffèrent en fonction des environnements et de l'utilisation ou non de ***Composer***.
- Typiquement avec ***Composer***, on ne versionne que les fichiers custom (modules, thèmes, profiles, librairies…). Tout le reste est récupérable via ***Packagist*** (ou autre dépôt compatible).
- Notons que ***Composer*** n'est pas utile en production car cela nécessiterait de refaire des tests.
- Par ailleurs lorsque l'on utilise ***Composer***, il est déconseillé d'installer de nouveaux modules ou thèmes via le back-office ou via ***Drush***. Toutes les librairies au sens large doivent être listées dans le fichier ***/composer.json***.



Exemple de fichier *.gitignore* avec *Composer*

```
# Ignore directories generated by Composer
/drush/contrib/
/vendor/
/web/core/
/web/modules/contrib/
/web/themes/contrib/
/web/profiles/contrib/
/web/libraries/

# Ignore sensitive information
/web/sites/*/settings.php
/web/sites/*/settings.local.php

# Ignore Drupal's file directory
/web/sites/*/files/

# Ignore SimpleTest multi-site environment.
/web/sites/simpletest

# Ignore files generated by PhpStorm
/.idea/
```

Déploiement via GIT

Commandes sur l'environnement **local**

```
drush config-export
git add .
git commit -m 'Modification des configurations.'
git push origin master
```

Commandes sur l'environnement **cible**

```
git pull origin master
drush updatedb
drush cache-rebuild
drush config-import
drush cache-rebuild
```

Système de configuration

Configuration vs Contenu

- L'architecture de *Drupal 8* est largement basée sur la notion d'entité.
- Chaque type d'entité est une structure permettant d'organiser l'information.
- Les différents types d'entités sont regroupées en 2 familles :
  - **entité de contenu** : noeud, taxonomie, utilisateur…
  - **entité de configuration** : vue, style d'image, format de texte…
- Le déploiement entre différentes instances d'un même site (DEV, STAGING, PROD…) concerne principalement la configuration, même si certains contenus peuvent être également déployés.

Config API vs State API

- On distingue :
  - la configuration exportable : ***Config API***
    - *Configuration simple* utilisant un formulaire unique (avec éventuellement des traductions) : information de base du site (titre, email…), performance (activation du cache, activation de l'agrégation CSS et JS…)…
    - *Configuration multiple* utilisant un formulaire (avec éventuellement des traductions) : les vues, les types de contenu, les vocabulaires de taxonomie…
  - l'état du site : ***State API***
    - date du dernier lancement des tâches planifiées, emplacement des fichiers en cache, mode maintenance…
- La *State API* regroupe des états propres à un environnement qui ne nécessitent pas d'être déployées.

Gestion de la configuration

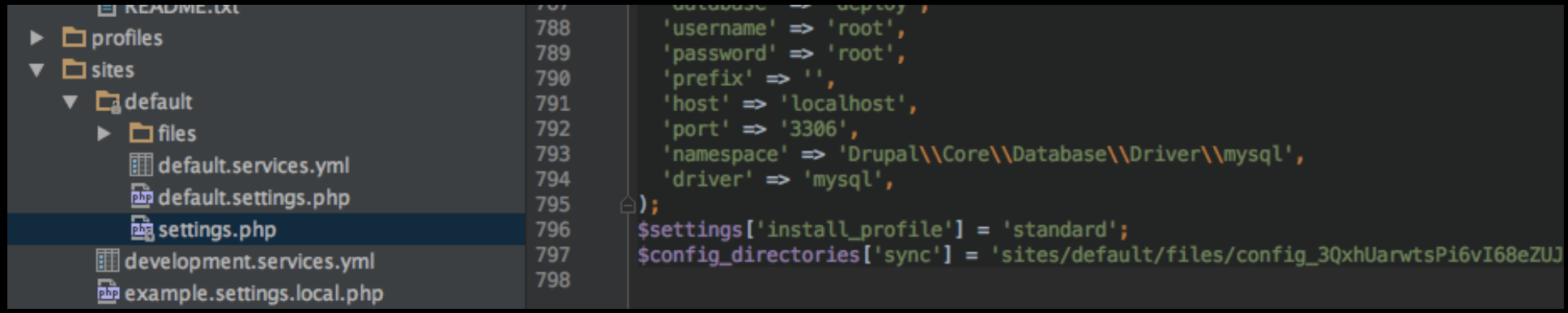
- Toute la **configuration** est **stockée en base**, pour des raisons de performances (table *config*).
- **Chaque configuration du site correspond à un fichier YAML**, par exemple les informations basiques du site sont contenues dans le fichier ***system.site.yml***.
- *Drupal 8* permet d'exporter/importer via le back-office ou en ligne de commande n'importe quelle configuration :
  - Export/import unitaire fichier par fichier.
  - Export/import complet sous forme d'archive comprenant tous les fichiers de configuration du site.

Organisation des fichiers

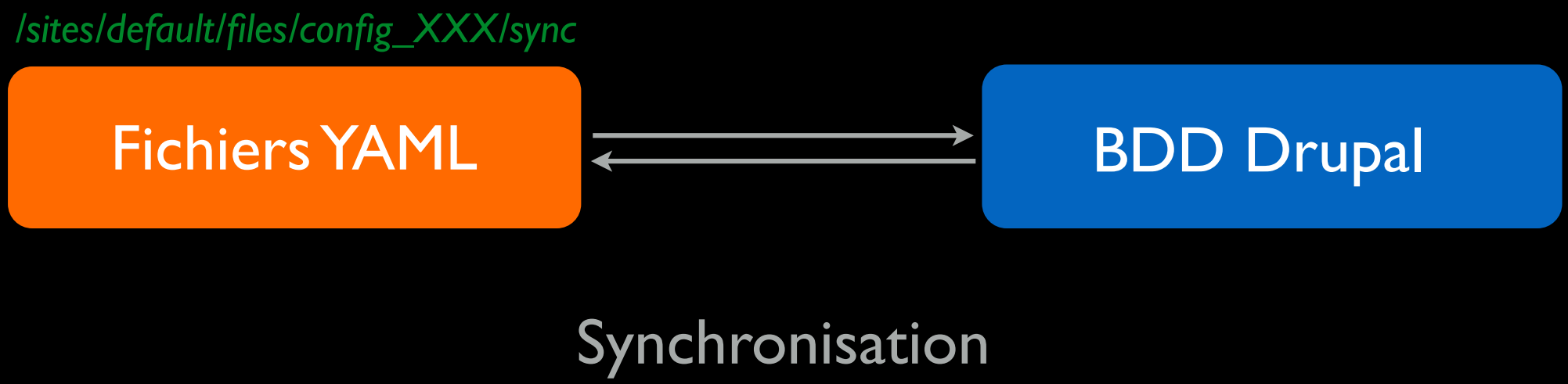
- Afin d'importer des changements de configuration, les fichiers correspondants doivent être placées dans le répertoire

/sites/default/files/config_XXX/sync

- Les fichiers sont donc « exposés », à condition de connaitre leurs URLs.
- Il est conseillé de déplacer ce dossier en dehors de la racine du site, afin qu'il ne soit pas accessible de l'extérieur.
- Il est possible de configurer ce répertoire dans le fichier */sites/default/settings.php*.
- ***Remarque** : il est possible également d'importer manuellement des configuration via le back-office.*



Synchronisation de configuration



Synchronisation de configuration

- Attention la configuration présente dans le répertoire de synchronisation doit comporter 2 fichiers obligatoirement :
  - ***core.extension.yml*** (modules activés)
  - ***system.site.yml*** (informations de base du site comme le titre)
- Il est fortement conseillé de placer la totalité de la configuration dans le répertoire de synchronisation, car sinon lors de l'import la configuration en base correspondante aux fichiers manquants est supprimée.
- Cet outil de synchronisation de la configuration est dédié au **déploiement d'un même site** entre différentes instances.

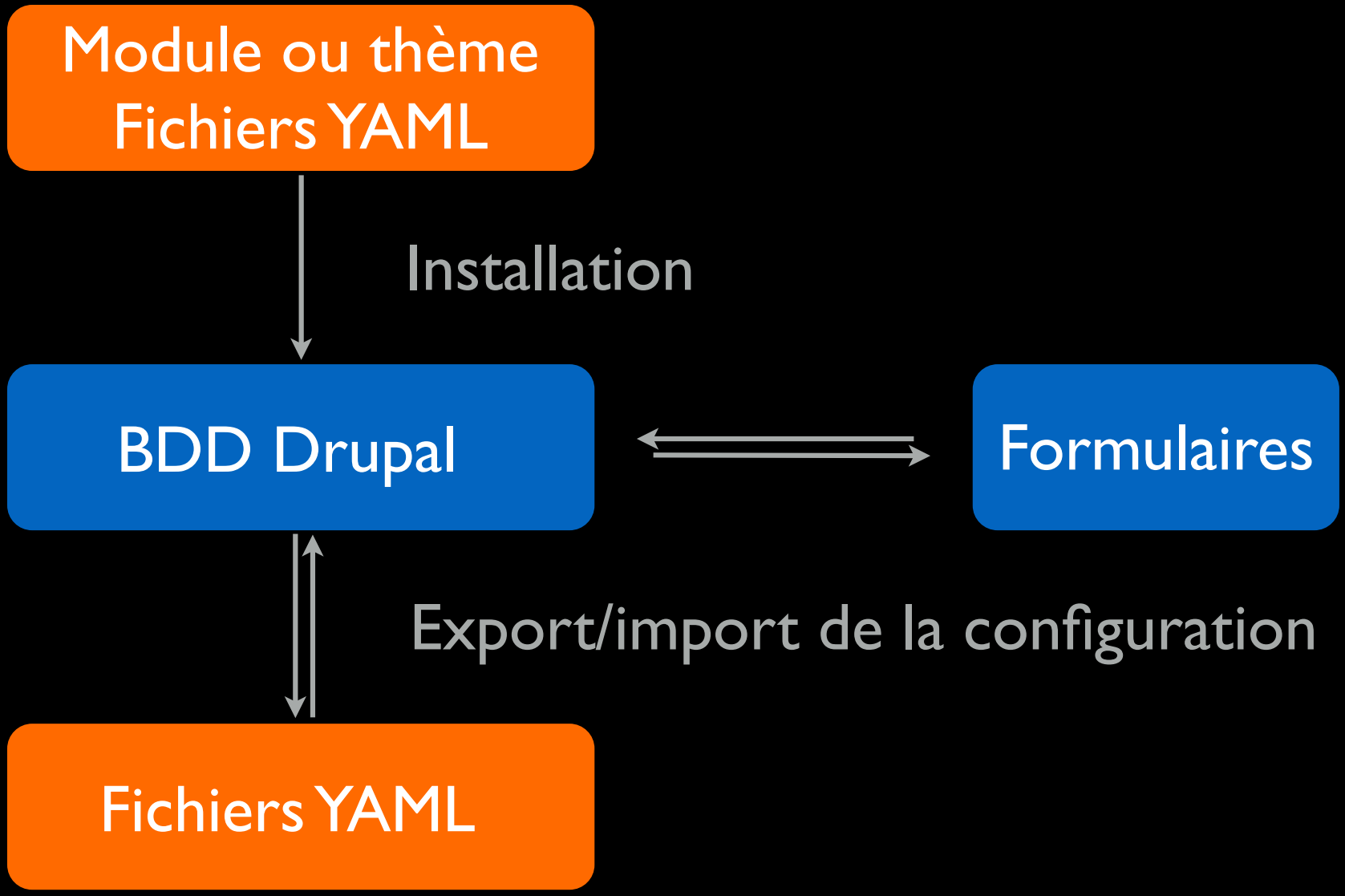
Export/import de la configuration

Configuration dans un module ou un thème

Configuration dans un module ou un thème

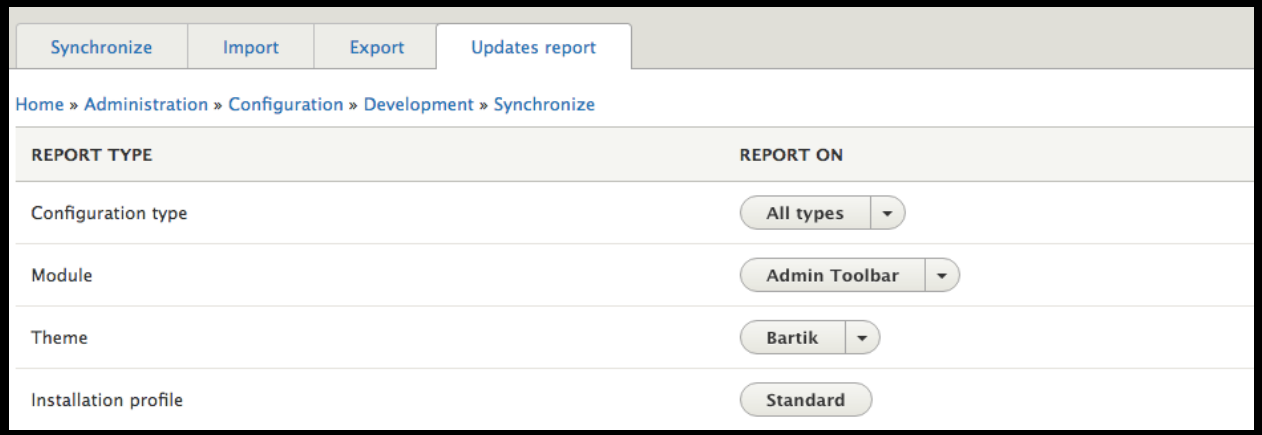
- **Lors de l'installation** d'un module/thème, *Drupal* importe toutes les configurations présentes dans le dossier ***/config/install*** et/ou ***/config/optional*** du module.
- Le premier dossier sert à la configuration définie par le module/thème ou bien le coeur de **Drupal**, tandis que le second est destiné à embarquer des configurations externes au module/thème, c'est à dire qui sont définies par des modules non requis.
- La configuration d'un module/thème est supprimée automatiquement lors de sa dés-installation.

Configuration par défaut



Configuration Update Base

- Par défaut **Drupal** ne peut pas mettre à jour une configuration embarquée dans un module. Les fichiers de configuration présents dans les répertoires */config/install* et */config/optional* ne sont pris en compte et chargés en base qu'au moment de l'installation.
- Le module ***Configuration Update Base*** permet de pallier cette limitation. Il est accompagné de son UI, ***Configuration Update Reports***. Pour découvrir les mises à jour de configuration des modules et thèmes installés aller sur *Admin > Configuration > Développement > Synchronisation de configuration > Updates report*.
- Ce module dépend du module ***Configuration Manager***.



Mise à jour d'une configuration par défaut

Configuration globale et configuration locale

Configuration locale et configuration globale

- En pratique on dispose de plusieurs instances du même site avec des **configurations** possiblement **différentes**.
- Par exemple un développeur installe des modules en plus pour ses tests et/ou son développement (modules ***Devel***, ***Devel Kint***…). Il ne peut donc pas directement remonter la configuration de son site local vers une branche commune.
- Certains modules ne devrait pas être activés en Production, par exemple les « UI » modules (**Field UI**, **Views UI**…), **Update Manager**…
- Il faut pouvoir faire des exports partiels de la configuration locale, tout en s'assurant de ne rien perdre par rapport à la branche commune.

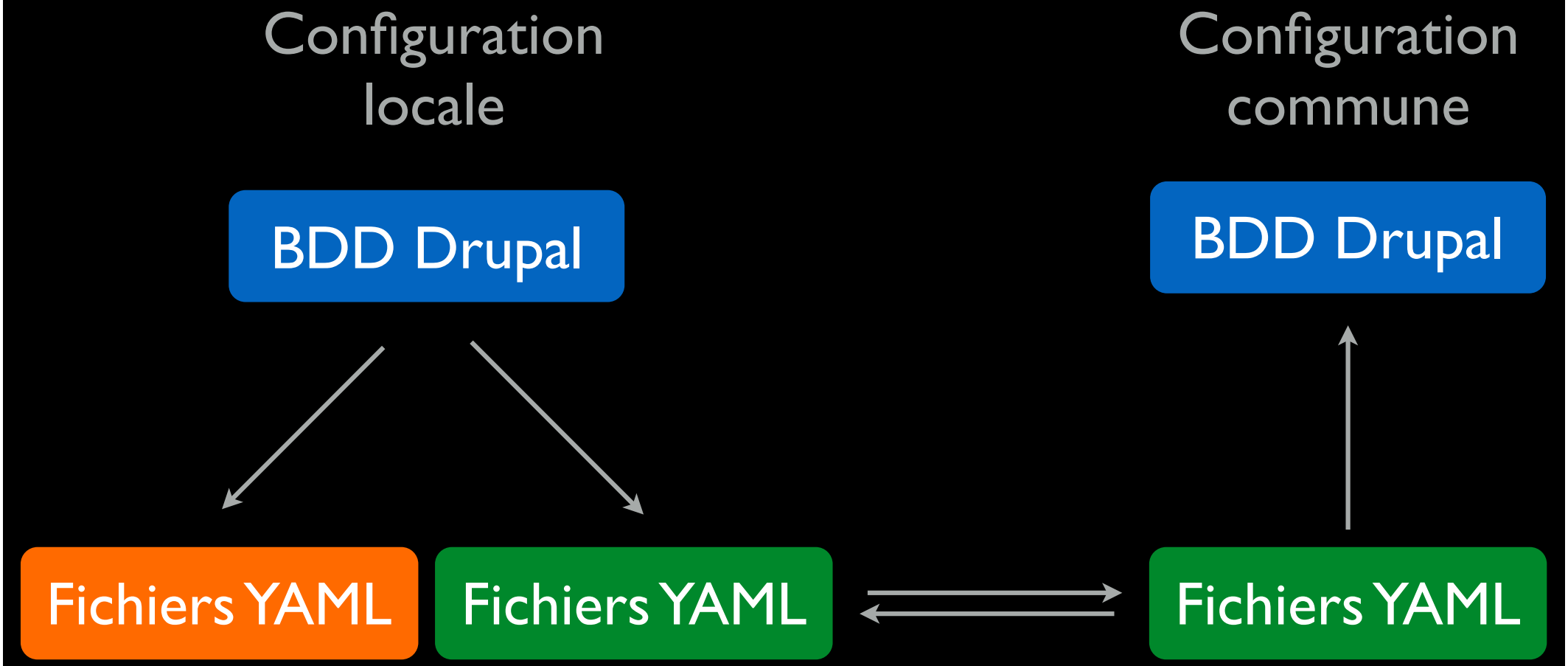
Configurations créées en Production

- Empêcher les changements de configuration en *Production* est une « bonne » idée, parfois malheureusement impossible à mettre en pratique.
- Dans certains cas, on est obligé d'autoriser la création de configuration directement en *Production*.
- C'est le cas par exemple des ***Webforms***. On peut considérer un site où on laisse la possibilité au webmaster de créer régulièrement de nouveaux formulaires. Or, avec **Drupal 8**, ces formulaires sont enregistrés en tant que configurations.

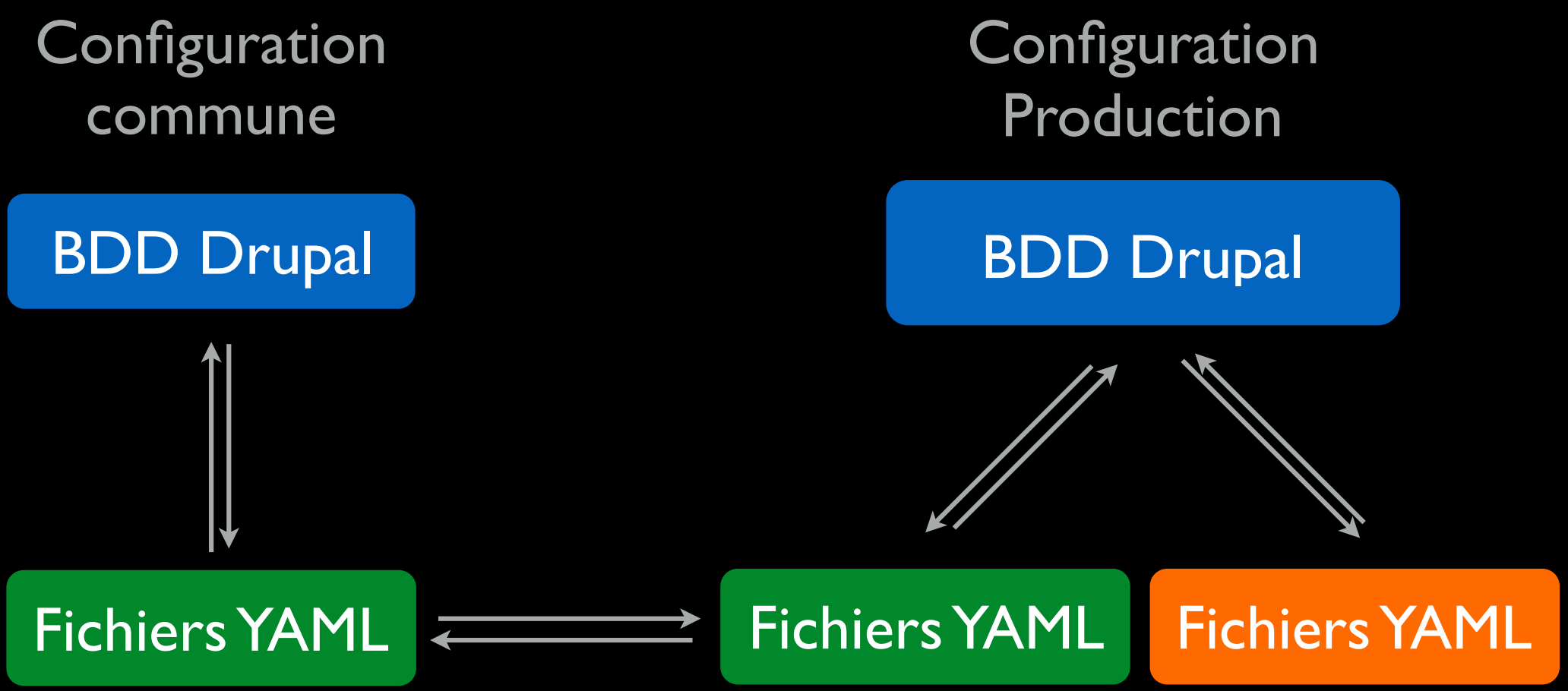
Configurations créées en Production

- Il ne faut pas risquer de voir ces formulaires supprimés ou écrasés au prochain déploiement.
- Il s'agit de **protéger les configurations** concernées à l'étape de l'import des configurations sur le site de destination.
- Avant d'importer les configurations en *Production*, il faut exporter toutes les configurations (incluant les nouvelles). On peut alors les ajouter au git (ou pas), suivant que l'on souhaite ou non les partager sur d'autres environnements.

Configuration locale et configuration globale



Configuration locale et configuration globale



Split et dossier d'export

- Le module ***Configuration Split*** *fonctionne* sur la base du module ***Config filter***, qui prend la main sur le canal normal d'export des configurations.
- Le dossier « sync » correspond à la configuration commune par défaut, celle que l'on souhaite partager entre les différents environnements. C'est le dossier déclarer dans le fichier ***/sites/default/settings.php*** (*$config_directories['sync']*).
- Les dossiers d'export additionnels sont déclarés dans chaque *split*. Le dossier correspondant n'est utilisé que si le *split* est actif.
- Les commandes *drush config-export (ou drush cex)* et *drush config-import (ou drush cim)* prennent en compte nativement les filtres de *Config Filter*.

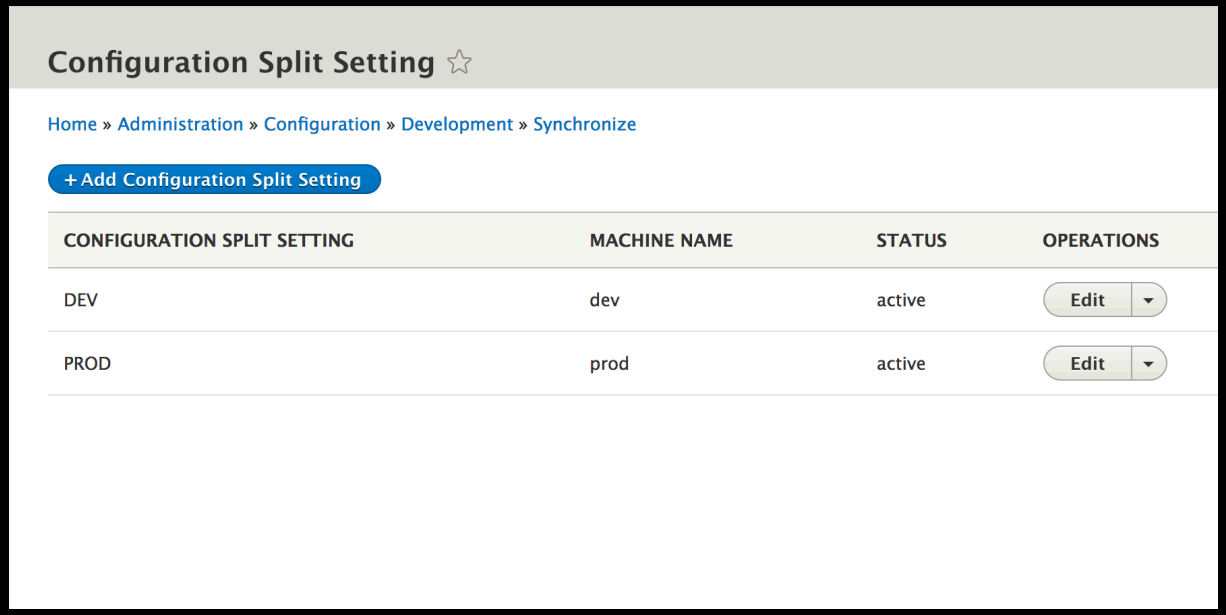
Split et dossier d'export

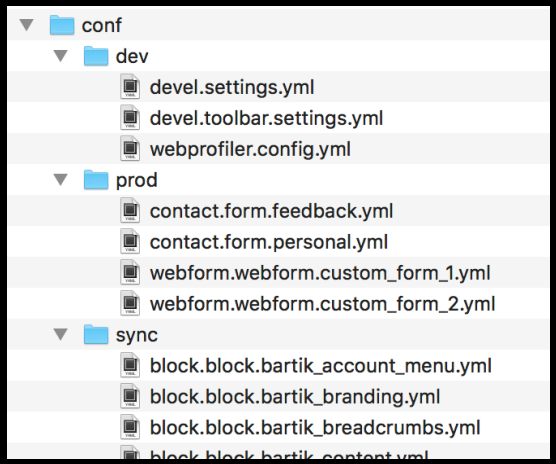
- Le module ***Configuration Split*** permet d'exporter la configuration totale de son site en direction de différents dossiers, en excluant certaines configurations.
- Il propose une interface permettant de créer des *split settings* définissant un dossier de destination et des règles d'exclusion :
  - liste des modules à ne pas activer.
  - liste des configurations à exclure de l'export principal.
- On a 2 types d'exclusion :
  - **configurations non partagées** : **Complete Split**. Généralement utilisée pour les développements.
  - **configurations différentes** entre les environnements : **Conditional Split**. Généralement utilisée pour la production.

Configuration des *Splits*

Exemple de *splits settings* possibles :

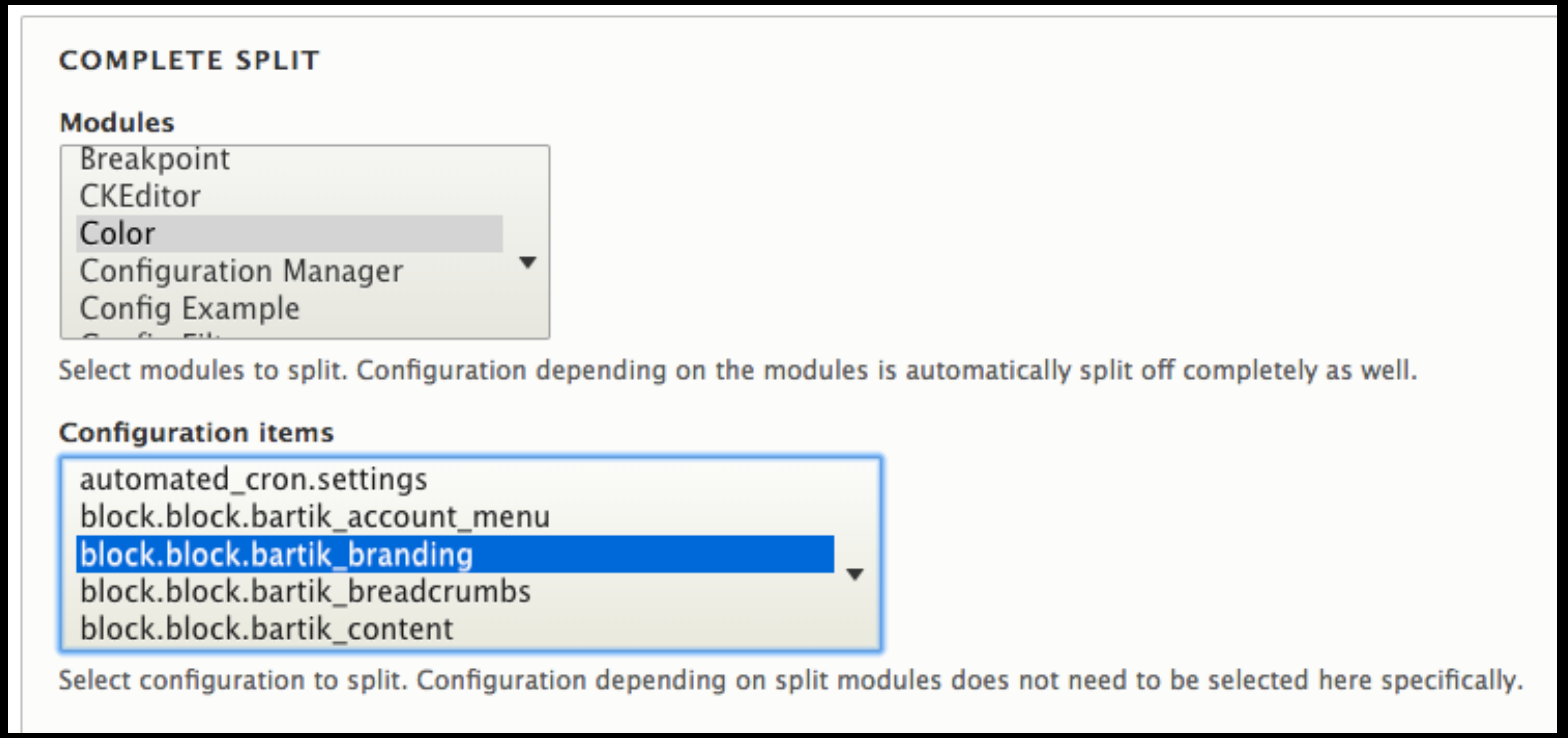
- *dev* pour les environnements locaux de développement.
- sync pour l'environnement de référence.
- *prod* pour l'environnement de *Production*.





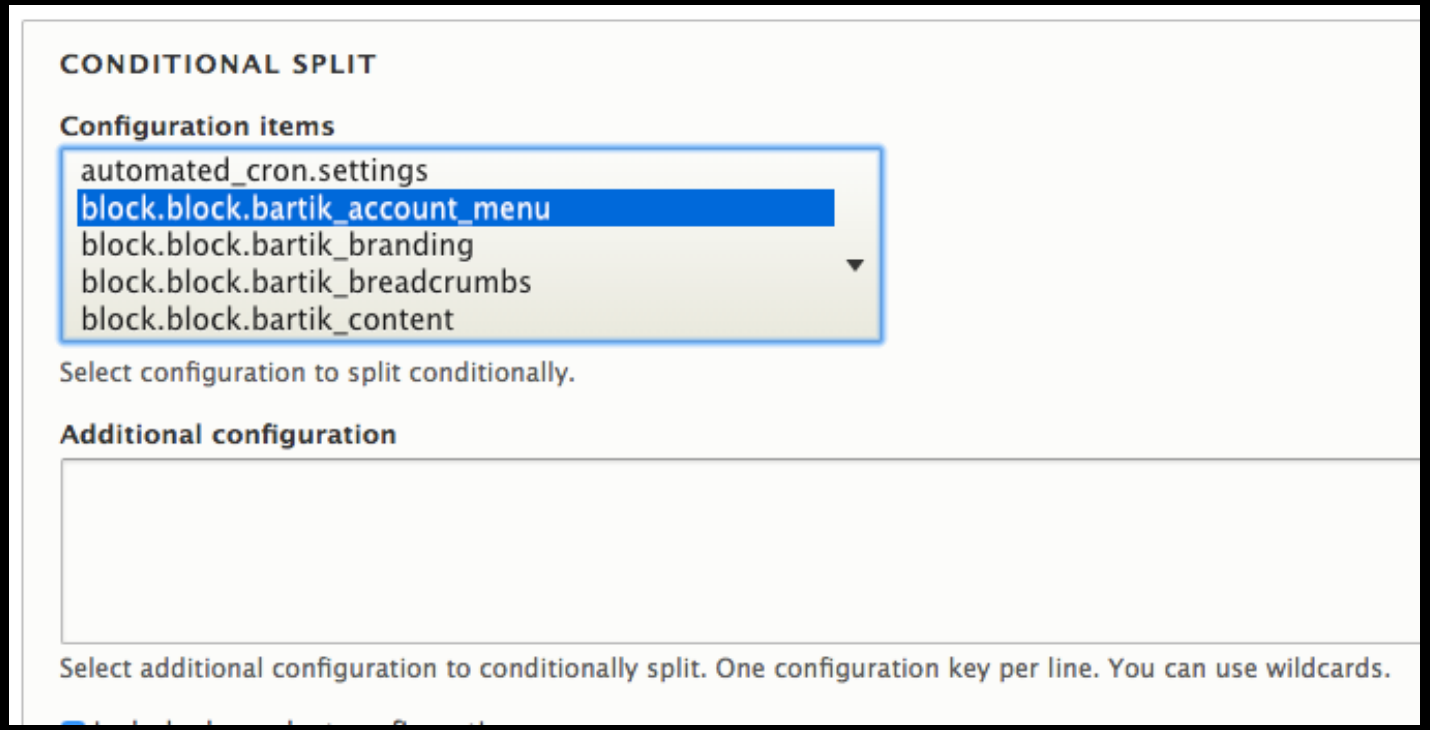
Complete Split (exclusion)

La section ***Complete Split*** définit les configurations en place dans l'environnement local de développement qui ne doivent pas être transférées en production, donc à **exclure de l'export**



Conditional Split (ignore)

- La section ***Conditional Split*** définit les configurations en place sur la production qui ne doivent pas être écrasées/supprimées à l'import.
- Elles seront exportées et fusionnées avec les configurations partagées (ou *globales*) à l'import.



Activation d'un *Split*

- Dans pour chacun des environnements, un *split* spécifique doit être activé.
- Souvent, il pourra être utile d'importer la configuration provenant de plusieurs dossiers/splits à la fois, en les « fusionnant ».
- L'activation d'un *split* ne doit jamais se faire via le back-office : lors du déploiement de la configuration correspondante, tous les environnement risqueraient de se voir attribués les mêmes *split*.
- On définit alors dans le fichier ***sites/default/settings.php*** spécifique à chaque environnement quel *split* est activé ou non.

```
$config['config_split.config_split.prod_split']['status']= TRUE;
```

Workflow basique

- Sur un environnement de dev, on **exporte** toute la configuration (dans 2 dossiers différents).
- Via ***Git*** on **partage** le dossier commun *sync* vers un environnement de staging.
- Sur la production, il faut tout d'abord exporter la configuration afin que les fichiers de *Conditional* soient à jour et qu'aucun ne manque. Puis on récupère les fichiers du staging via ***Git***. Seul le dossier *sync* est mis à jour. Enfin on **importe** toute la configuration (fichiers YAML communs et spécifiques à la production).

```
Differences of the active config to the export directory:

+------------+-------------------------------+-----------+
| Collection | Config                        | Operation |
+------------+-------------------------------+-----------+
|            | dblog.settings                | Create    |
|            | block.block.bartik_help       | Create    |
|            | block.block.seven_help        | Create    |
|            | views.view.watchdog           | Create    |
|            | menu_ui.settings              | Create    |
|            | tour.tour.views-ui            | Create    |
|            | core.extension                | Update    |
|            | config_split.config_split.dev | Update    |
+------------+-------------------------------+-----------+

 The .yml files in your export directory (../config-deploy/sync) will be dele
 >

 [success] Configuration successfully exported to ../config-deploy/sync.
MacBook-Air-de-Romain:deploy romainjarraud$ drush cim
+------------+-------------------------------+-----------+
| Collection | Config                        | Operation |
+------------+-------------------------------+-----------+
|            | core.extension                | Update    |
|            | config_split.config_split.dev | Update    |
|            | tour.tour.views-ui            | Delete    |
|            | menu_ui.settings              | Delete    |
|            | views.view.watchdog           | Delete    |
|            | block.block.seven_help        | Delete    |
|            | block.block.bartik_help       | Delete    |
|            | dblog.settings                | Delete    |
+------------+-------------------------------+-----------+

 Import the listed configuration changes? (yes/no) [yes]:
 >

 [notice] Synchronized extensions: uninstall tour.
 [notice] Synchronized extensions: uninstall views_ui.
 [notice] Synchronized extensions: uninstall menu_ui.
 [notice] Synchronized extensions: uninstall help.
 [notice] Synchronized extensions: uninstall history.
 [notice] Synchronized extensions: uninstall dblog.
 [notice] Synchronized extensions: uninstall admin_toolbar_tools.
 [notice] Synchronized configuration: update config_split.config_split.dev.
 [notice] Finalizing configuration synchronization.
```

Export partiel de la configuration

Surcharger la configuration

Surcharger la configuration

- Pour contrôler l'état des configurations du site sur les différents environnements, on peut juger risqué de ne se fier qu'aux fichiers *YAML* comme source et leur correct filtrage par les *config split/filters*. Ceux-ci peuvent avoir été mal paramétrés ou modifiés par un des collaborateurs à l'export et la livraison des configurations.
- Aussi, il est possible, même si une variable de configuration a été importée qui n'aurait pas due l'être, de «forcer» sa valeur durant le temps d'exécution.

Surcharger la configuration

- Il est possible de surcharger n'importe quelle configuration dans le fichier */sites/default/settings.php* (ou équivalent).
- Ces éventuelles surcharges n'apparaissent pas dans l'interface du site et ne sont pas exportées dans les fichiers de configuration YAML. Ce sont des configurations locales à l'environnement.

```
617      * configuration values in settings.php will not fire any of the configuration
618      * change events.
619      */
620     $config['system.site']['name'] = 'My Drupal site';
621     # $config['system.theme']['default'] = 'stark';
622     # $config['user.settings']['anonymous'] = 'Visitor';
623
624     /**
625      * Fast 404 pages:
```

Surcharger la configuration

- Le fichier ***/sites/default/settings.php*** est normalement proprement spécifique à chaque environnement, comportant des configurations uniques (ex: accès à la base de données).
- On peut néanmoins mettre à disposition des fichiers déclarant des surcharges de configurations spécifiques adaptées à chaque type d'environnement (local, preprod, prod, etc…), permettant aux collaborateurs d'y accéder et de les modifier à divers fins.
  - Quand le développement local sollicite un jeu de configurations complexes difficile à gérer individuellement (gestion des splits, configuration de solutions de debug, endpoints de webservices «de test» différents de la prod, backend de cache spécifique, etc…). Ce partage sera d'autant puissant s'il est couplée à un environnement mutualisé *par virtualisation* ou de *containerisation* (***Vagrant*** ou ***Docker***).
  - Quand on veut gérer tout ou partie des configurations de la pré-production ou production depuis le dépôt GIT si un cherche le déploiement le plus automatique possible (attention, risqué!).

Surcharger la configuration

- Le dépôt GIT peut servir à versionner et rendre accessible différents fichiers ***settings.php***.
- Manuellement, dans chacun des environnements, il s'agira alors d'inclure un des fichiers à disposition depuis le fichier ***/sites/default/settings.php*** principal. On peut ainsi avoir différents fichiers :
  - ***/sites/default/settings.local-dev.php***
  - ***/sites/default/settings.preprod-example.php***
  - ***/sites/default/settings.prod-example.php***

Exemple de surcharges

/sites/default/settings.php

```
  'namespace' => 'Drupal\\Core\\Database\\Driver\\mysql',
  'driver' => 'mysql',
);

# Ajout des configurations partagées
if (file_exists(__DIR__ . '/settings.local.php')) {
  include __DIR__ . '/settings.local.php';
}
```

/sites/default/settings.local.php (environnement de développement local)

```
$settings['twig_debug'] = TRUE;
$settings['twig_auto_reload'] = TRUE;
$settings['twig_cache'] = FALSE;
$settings['cache']['bins']['render'] = 'cache.backend.null';
$settings['cache']['bins']['dynamic_page_cache'] = 'cache.backend.null';
$settings['cache']['default'] = 'cache.backend.memcache_storage';
$settings['memcache_storage']['debug'] = FALSE;
$settings['memcache_storage']['extension'] = 'Memcached';

$config['system.performance']['cache']['page']['max_age'] = '20000';
$config['system.logging']['error_level'] = ERROR_REPORTING_DISPLAY_VERBOSE;
$config['system.performance']['css']['preprocess'] = FALSE;
$config['system.performance']['js']['preprocess'] = FALSE;
$config['devel.settings']['devel_dumper'] = 'kint';
$config['user.role.anonymous']['permissions'][10000] = 'access devel information';
$config['user.role.authenticated']['permissions'][10000] = 'access devel information';
$config['config_split.config_split.dev_split']['status']= TRUE;
```

Mode de lecture seule

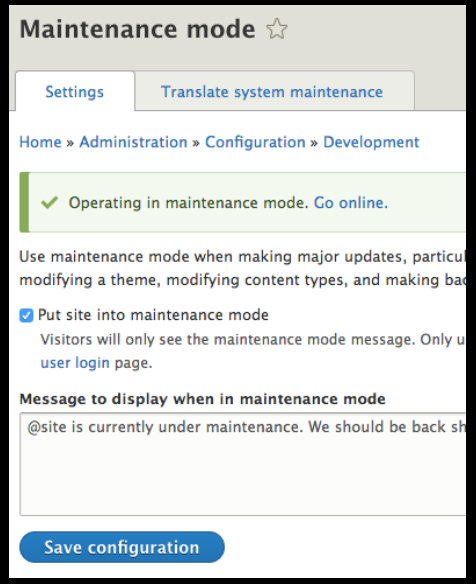
Mode de lecture seule

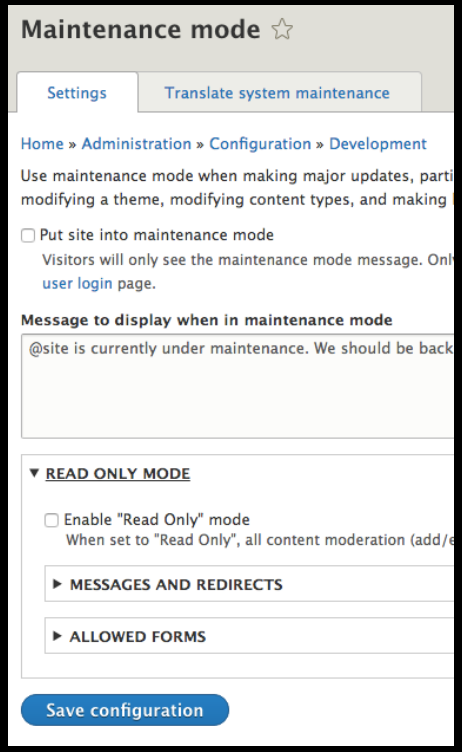
- Lors d'un déploiement du site, il est vivement conseillé de mettre le site en mode de maintenance afin de s'assurer qu'il n'y a plus d'accès en écriture sur la base de données.
- En mode maintenance, le site n'est plus accessible (sauf de ceux ayant la permission *Accèder au site en mode maintenance*). Seule une page affichant un message d'avertissement est visible.
- Le module ***Read Only Mode*** permet de bloquer la création de contenu et la modification de la configuration du site. Par défaut aucun formulaire n'est accessible (sauf pour se connecter).
- Le module ***Configuration Read-only*** permet lui de ne bloquer que la configuration du site. Aucun formulaire du back-office n'est accessible. Noter que cela nécessite une activation dans le fichier de settings ***/sites/default/settings.php*** :

```
$config['config_readonly'] = TRUE;
```

Module *Read Only Mode*

- Pendant un déploiement, il est nécessaire de passer le site en **mode maintenance**. Pour ce faire aller sur *Admin > Configuration > Développement > Mode maintenance*.
- Le module ***Read Only Mode*** ajoute des options à ce formulaire : message à afficher ou redirection, et liste des formulaires autorisés.
- Le mode de lecture seule est indépendant du mode de maintenance. En pratique on utilise soit l'un soit l'autre.





Remarques

Gestion des schemas de définitions des entités

- La commande ***drush entity-update*** permet, en cas de changement dans les définitions des champs d'un type d'entité, de remettre en cohérence le schéma avec le champ de la table concernée en base de donnée, et donc les données des entités enregistrées. Cette commande permet une remise en cohérence automatique mais « forcée » qui pourra occasionner des anomalies et des régressions importantes.
- Elle est potentiellement dangereuse. Ce type de modification devrait être proprement géré dans un ***hook_entity_update()***. Le changement de la définition peut impliquer un traitement sur-mesure, par exemple la conversion des données des entités existantes.
- ***Drush entity-update*** n'a donc d'intérêt que pour les modules custom en phase de développement et dans la mesure ou elle doit toujours imposer des vérifications.

Fonctions *hook_update_N()*

- Le ***hook_update_N()*** du fichier *.install* d'un module peut servir à déclarer l'ensemble des ajouts, suppressions et modifications d'entités de contenu (nœuds, termes de taxonomies, etc.) qu'il est nécessaire de déployer.
- Il est possible de répartir ces updates dans différents modules et/ou créer un module spécifique (ex: *project_updates*) où les divers updates sont centralisés.

/modules/custom/project_updates/project_updates.install

```php
/**
 * Implements hook_update_N
 */
function project_updates_update_8003() {
    $em = \Drupal::entityTypeManager()->getStorage('taxonomy_term');
    $new_terms = [
        ['name' => 'Finance', 'vid' => 'categories'],
        ['name' => 'Sport', 'vid' => 'categories'],
    ];
    foreach($new_terms as $new_term){
        $newEntity = $em->create($new_term);
        $em->save($newEntity);
    }
}
```

- Cette solution impose de coder chaque traitement, ce qui demande un travail assez fastidieux, surtout en cas d'entités ou règles de synchronisation nombreuses et complexes.

Les étapes d'un déploiement

Remarques

- En fonction de la nature du projet, des habitudes de l'équipe de développement et de l'environnement cible (serveur de développement, d'intégration, recette, production, etc.) les commandes pourront varier.
- ***Composer update*** doit être préférée à ***drush pm-update***, car la commande ne pourra pas réaliser l'ensemble des mises à jour si les modules contrib ont été déclarés et intégrés en dehors de **Composer**.
- La mise à jour du code des bibliothèques **PHP**, modules contrib et cœur de **Drupal** via ***composer update*** n'est pas nécessairement à réaliser à chaque déploiement (encore moins à chaque commit). Elle peut impliquer des modifications majeures du code, donc des risques de régression imposant normalement une nouvelle batterie de tests.

Les étapes d'un déploiement

- **Environnement local** :
  - Mettre à jour le code contrib (**Composer**).
  - Mettre à jour la base de donnée (**Drush**).
  - Exporter la configuration (**Drush**).
  - Commiter le code custom et la configuration (**GIT**).
- **Environnement cible** :
  - Passer en mode maintenance (**Drush**).
  - Exporter la configuration (**Drush**) si utilisation du module ***Config Split***.
  - Récupérer le code custom et la configuration (**GIT**).
  - Mettre à jour le code contrib (**Composer**).
  - Mettre à jour la base de donnée (**Drush**).
  - Vider les caches (**Drush**).
  - Importer la configuration (**Drush**).
  - Vider les caches (**Drush**).
  - Désactiver le mode maintenance (**Drush**).

```
#!/usr/bin/env bash

# Passage en mode maintenance.
drush sset system.maintenance_mode 1
echo Maintenance mode enabled.
drush cr

# Mise à jour du code.
git pull origin master
composer install --no-dev
drush cr

# Mise à jour de la base.
drush updb
drush cr

# Mise à jour des schemas des types d'entités.
drush entup
drush cr

# Export des configs de prod.
drush csex prod -y

# Import des configs.
drush cim -y
drush cr

# Ajout des configs de prod au master.
git add config/prod
git commit -m 'Ajout des configs de prod.'
git push origin master

# Sortie du mode maintenance.
drush sset system.maintenance_mode 0
drush cr
echo Site is online.
```

Merci !