DÉPLOIEMENT ET BACKUP AUTOMATIQUE D'UNE APPLICATION SYMFONY 4

Présentation

Vous avez enfin fini le développement de votre **site web local** avec Symfony, mais voilà vous avez besoin de le **déployer sur un serveur web de production** mais aussi d' **automatiser le déploiement des mises à jour**. Si c'est le cas alors vous êtes sur le bon article.

Avant de vous montrer comment automatiser les processus, laissez-moi d'abord vous expliquer ce que j'entends par un serveur web de production.

C'est tout simplement un serveur qui héberge votre site web et reste souvent sollicité par des internautes.

La pipeline de déploiement reste la même que ça soit pour une machine que vous louez directement chez un **hébergeur web** ou un serveur que vous gérez vous-même.

Je vais donc vous décrire à travers cet article comment je procède pour déployer et mettre à jour automatiquement mes applications Symfony sur mes serveurs de production.

Vous êtes prêt ? Si oui alors commençons !

Les prérequis

- Un **accès ssh** sur la machine de production est obligatoire !

- Pour mieux comprendre cet article, il est préconisé de connaître au minimum les bases de :
    - **Symfony 4**
    - **Makefiles**
    - **rsync**

Je tâcherai tout de même de vous expliquer les processus que j'ai utilisés ainsi que mon code source. Je vous recommande tout de même de connaître au moins les bases des technologies citées plus haut pour vous faciliter la compréhension de cet article.

Pourquoi le choix du Makefile ?

Make est un outil puissant permettant, entre autres, d'**automatiser la compilation d'un projet**. Il qui est habituellement utilisé pour la compilation des programmes codés en C. Cependant, comme vous vous en doutez, son usage ne se limite pas à cela, on peut aussi l'utiliser pour nos projets web.

L'avantage de make c'est qu'il est installé par défaut sur les plateformes UNIX et qu'il est aussi facilement installable sur Windows. Windows vous offre également l'option du WSL. Cela vous donne accès à un shell bash natif.

Pour ce qui est des Makefiles, ce ne sont que des fichiers utilisés par l'outil make pour exécuter un ensemble de commande sous forme d'actions.

À la différence d'un simple script shell, make **exécute les commandes seulement si elles sont nécessaires**. Le but est d'arriver à un résultat final sans nécessairement refaire toutes les étapes.

Pourquoi le choix du rsync ?

*"Ok ok, on a compris le Makefile permet de **lister les actions à exécuter** qui seront ensuite exécuter par l'outil make, mais qu'en est-il du rsync ?" ?*

rsync (Remote syncronization) est un outil de **synchronisation de fichiers** en ligne de commandes. Il fonctionne de manière **unidirectionnelle**, c'est-à-dire qu'il va copier les fichiers d'une source (locale ou distante) vers une destination (locale ou distante) et ne prendra en compte que les **modifications** faites sur les fichiers sources.

L'avantage majeur de rsync est qu'il est lui aussi installé par défaut sur les plateformes UNIX mais c'est surtout qu'il ne va transférer que les fichiers modifiés ce qui permet d'avoir un sacré gain de temps !

Voici quelques **options de rsync** qu'on utilisera dans notre Makefile

Option	Description
`-a`	Équivalant de l'option `-rlptgoD`. C'est un moyen rapide de dire que vous voulez utiliser la récursion et presque tout conserver (date, permission, etc ...)
`-v`	Activer le mode verbose
`--exclude`	Ignorer la synchronisation d'un fichier ou dossier ou d'un pattern (exemple *.zip pour ignorer tous les fichiers zip
`--delete`	**Supprimer les fichiers non présents dans le dossier source !**

Les actions du Makefile

Etape 1 : le Backup

En cas de mise à jour, il est important de **créer un système de sauvegarde** de l'application hébergée sur le serveur distant. Cette manipulation nous permettra un possible **retour en arrière** en cas de soucis.

Pour simplifier les choses on va plutôt se poser la question suivante : *"Lors d'une mise en place d'un système sauvegarde qu'est-ce qu'on peut **ignorer** comme dossiers ou fichiers pour une application Symfony ?"*

Déjà si vous utilisez git (ce que je vous recommande) alors voici à quoi va ressembler généralement l'arborescence de votre application :

```
|___ bin
|___ composer.json
|___ composer.lock
|___ config
|___ .env
|___ .git
|___ .gitignore
|___ phpunit.xml.dist
|___ public
|___ src
|___ symfony.lock
|___ templates
|___ translations
|___ var

|___ vendor
```

Par défaut, parmi tous les dossiers existants de l'application, seuls les dossiers **var** et **vendor** restent volumineux. Dans le cas d'une sauvegarde il n'est pas nécessaire de les conserver car ils peuvent être regénérés automatiquement, sauf si vous souhaitez garder les logs de votre application symfony qui sont dans le dossier `var`, puisque ces derniers peuvent contenir des informations très importantes).

vendor est le dossier où Composer stocke tous les paquets et les dépendances. Ce dossier peut être régénéré grâce à la commande suivante :

```
composer install
```

Quant au dossier **var**, il permet de mettre en cache votre application, mais ce dossier contient aussi les **logs** de votre application. Les logs restent super-

importants, surtout en prod car ça reste le seul moyen de vérifier s'il existe des erreurs de code.

Plus tard, quand je vous dévoilerai mon Makefile, vous verrez que je sépare la partie sauvegarde en deux parties. Je vous expliquerai les raisons de ce choix plus bas dans cet article.

Etape 2 : le déploiement

Une fois que vous aurez sauvegardé les sources distantes de votre application l'étape suivante sera le déploiement de vos changements de votre environnement de test vers votre environnement de production.

Avant de procéder à l'étape de déploiement, il faut absolument se poser la question suivante :*"Lors d'une mise en place d'un système de déploiement qu'est-ce qu'on peut* ***ignorer*** *comme dossiers ou fichiers pour une application Symfony ?"*

Pour commencer je peux déjà me débarrasser tout ce qui est en rapport avec à git, dès lors je vais ignorer le dossier `.git` et tous les fichiers `.gitignore`.

De la même façon, j'ignore aussi le fichier `.env` car ma configuration dans l'environnement de test est différente de celle de mon environnement de production (base de données différente, serveur de messagerie différent, etc ..). Ainsi, le fichier `.env` de prod sera créé manuellement avec la bonne configuration directement sur le serveur de production. Cette précaution, m'évite de stocker des informations telle que le mot de passe ma base de données sur mon environnement de test (pratique quand votre environnement de test est votre pc perso et qu'on oublie de verrouiller pas son pc)

Parallèlement j'ignore aussi le dossier `var` car ce dernier ne contient que le cache et les logs de mon environnement de test et donc non nécessaire sur mon serveur de

production.

Par contre je garde le dossier `vendor` car déjà sur le serveur distant je ne souhaite en aucun cas installer des outils que je peux éviter (je pense notamment à composer). Ensuite sur votre environnement de test vous pouvez changer la variable `APP_ENV` du fichier `.env` en `prod` et par la suite tester vos paquets téléchargés sur cette nouvelle valeur localement. Une fois que vous êtes satisfait du résultat alors vous pouvez lancer la phase de déploiement et ainsi vous assurez que les paquets que vous envoyez sont bel et biens compatibles avec une **configuration de production**.

Le premier déploiement sera certes lent à cause de l'envoi complet du dossier `vendor` mais par la suite ça sera beaucoup plus rapide sur les futurs déploiements .

Pour finir j'ignore aussi les DataFixtures car elles sont utiles que pour créer des fausses données dans ma base de données pour mes tests, de plus le bundle reste activé qu'en mode dev.

> **Attention**
>
> Si vous stockez dynamiquement des fichiers médias (du style photos, vidéos, etc ...) alors il faut impérativement les ignorer à fin de ne pas les supprimer sur votre serveur de prod quand vous lancerez votre déploiement !

Création du Makefile

Après beaucoup de théories, passons à la pratique !

Les variables globales

Premièrement je commence par créer dans la racine de mon application un fichier **Makefile** et je déclare dedans les variables que je serai susceptible d'utiliser sur plusieurs de mes actions.

```
remote_ssh:=[UTILISATEUR]@[HOST]
remote_src:=[SOURCE_DISTANTE]

backup_location:=[DOSSIER_LOCAL_DE_BACKUP)


# Couleurs pour rendre jolie l'affichage
COM_COLOR   = \033[0;34m
OBJ_COLOR   = \033[0;36m
OK_COLOR    = \033[0;32m
ERROR_COLOR = \033[0;31m
WARN_COLOR  = \033[0;33m
NO_COLOR    = \033[m
```

Je commence d'abord par déclarer une variable nommée `remote_ssh` qui aura comme valeur l'accès ssh à votre serveur distant.

Ensuite je crée une variable nommée `remote_src` qui prendra comme valeur le chemin complet de votre application sur la machine distante.

Enfin je déclare une variable nommée `backup_location` qui portera comme valeur le chemin complet de sauvegarde de ma machine locale.

La sauvegarde des fichiers lourds

Malgré le fait que j'ai choisi d'ignorer deux dossiers lourds à savoir **vendor** et **var**, il n'en reste pas moins que d'autres dossiers soient eux aussi à leur tour très dense. Je pense notamment aux images (dans mon cas ça sera les images et les documents PDF). Pour ce type de fichier je préfère les stocker dans un seul et même dossier localement. Ce choix va me permettre de gagner de la place et du stockage.

Ce qui va donner dans mon Makefile

```
backup_media:=$(backup_location)/media/

mediaBackup: ## Sauvegarde des fichiers lourds (images et PDF) en incrémental dans u
    @echo -e "\n$(WARN_COLOR)- Récupération des $(WARN_COLOR)medias$(WARN_COLOR) du 
    @rsync -auv \
            --delete \
        $(remote_ssh):$(remote_src)/public/images $(backup_media)
    @rsync -auv \
            --delete \
            $(remote_ssh):$(remote_src)/public/documents $(backup_media)
```

Mes images et mes documents PDF seront automatiquement stockés localement sur le dossier de **backup/media**. Si les images et les documents sont déjà sauvegardés localement alors l'outil make ignorera cette étape ce qui nous permettra de gagner du temps (essentiel pour déployer une mise à jour critique).

La sauvegarde des sources

En ce qui concerne les sources de l'application (config + code sources sans les images) j'ai choisi de les sauvegarder dans un nouveau dossier crée automatiquement à chaque sauvegarde. Le nom du dossier sera composé de la date et de l'heure de sauvegarde. Pour rappel, je ne sauvegarde pas les images et mes documents PDF dans ces nouveaux dossiers car je le fais déjà fait pour l'action **`mediaBackup`** de mon Makefile et j'ignore aussi les dossiers **var** et **vendor** pour les raisons expliquées plus haut.

Ce qui nous donne dans le fichier Markdown :

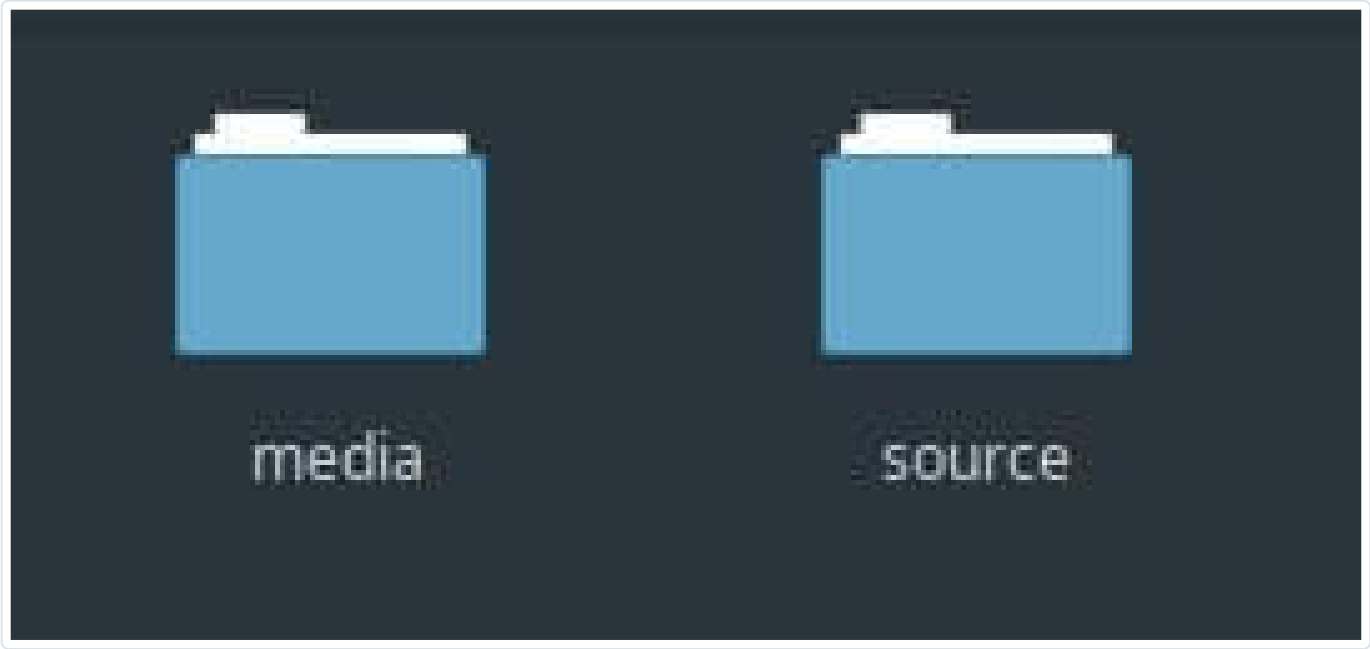
```
date_backup=$(shell date +'%d-%m-%Y__%H-%M-%S')
backup_source:=$(backup_location)/source/$(date_backup)/

backup: mediaBackup ## Crée un nouveau dossier et sauvegarde dedans les sources
 @echo -e "\n$(WARN_COLOR)- Récupération des $(WARN_COLOR)sources$(WARN_COLOR) du se
 @rsync -auv \
 --exclude '/vendor' \
 --exclude '/var/cache' \
```

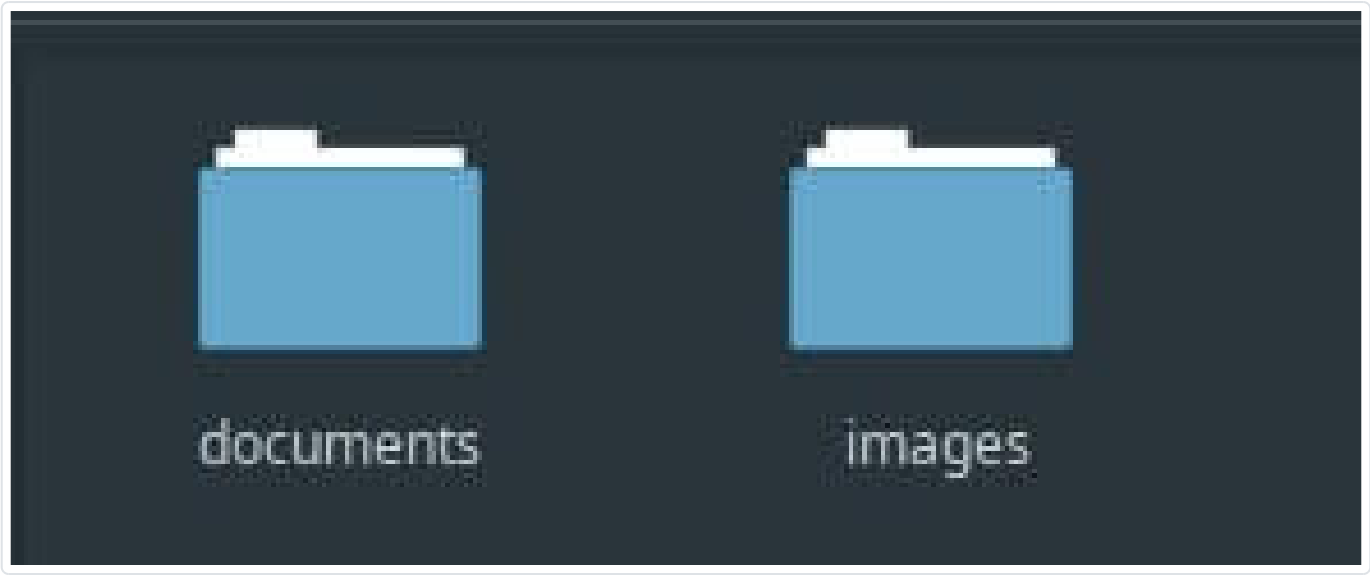
```
    --exclude '/public/media' \
    --exclude '/public/images' \
    --exclude '/public/documents' \
       $(remote_ssh):$(remote_src) $(backup_source)
```

Voici un aperçu des dossiers suivants :

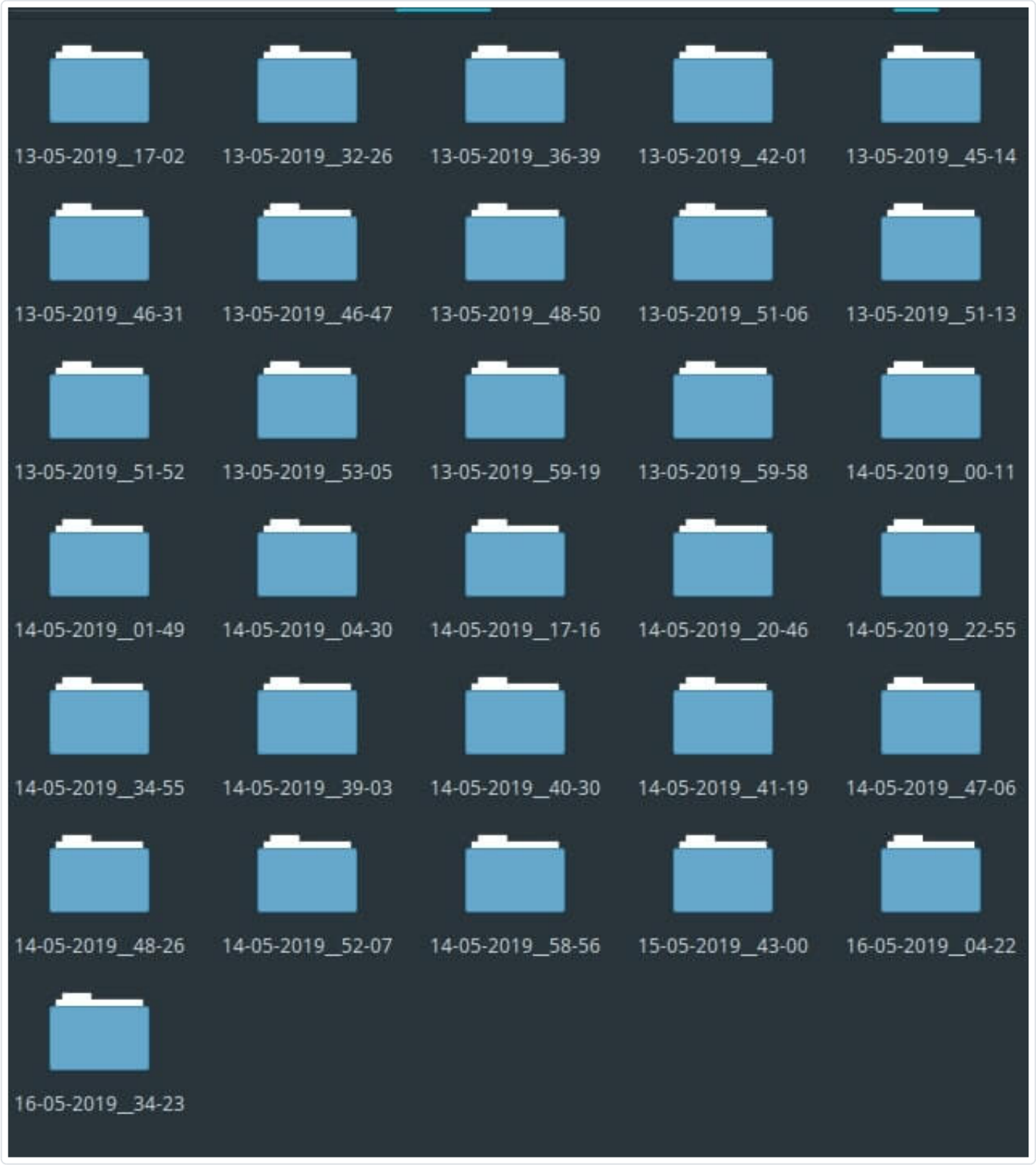
- **Backup :**



- **Backup/media :**



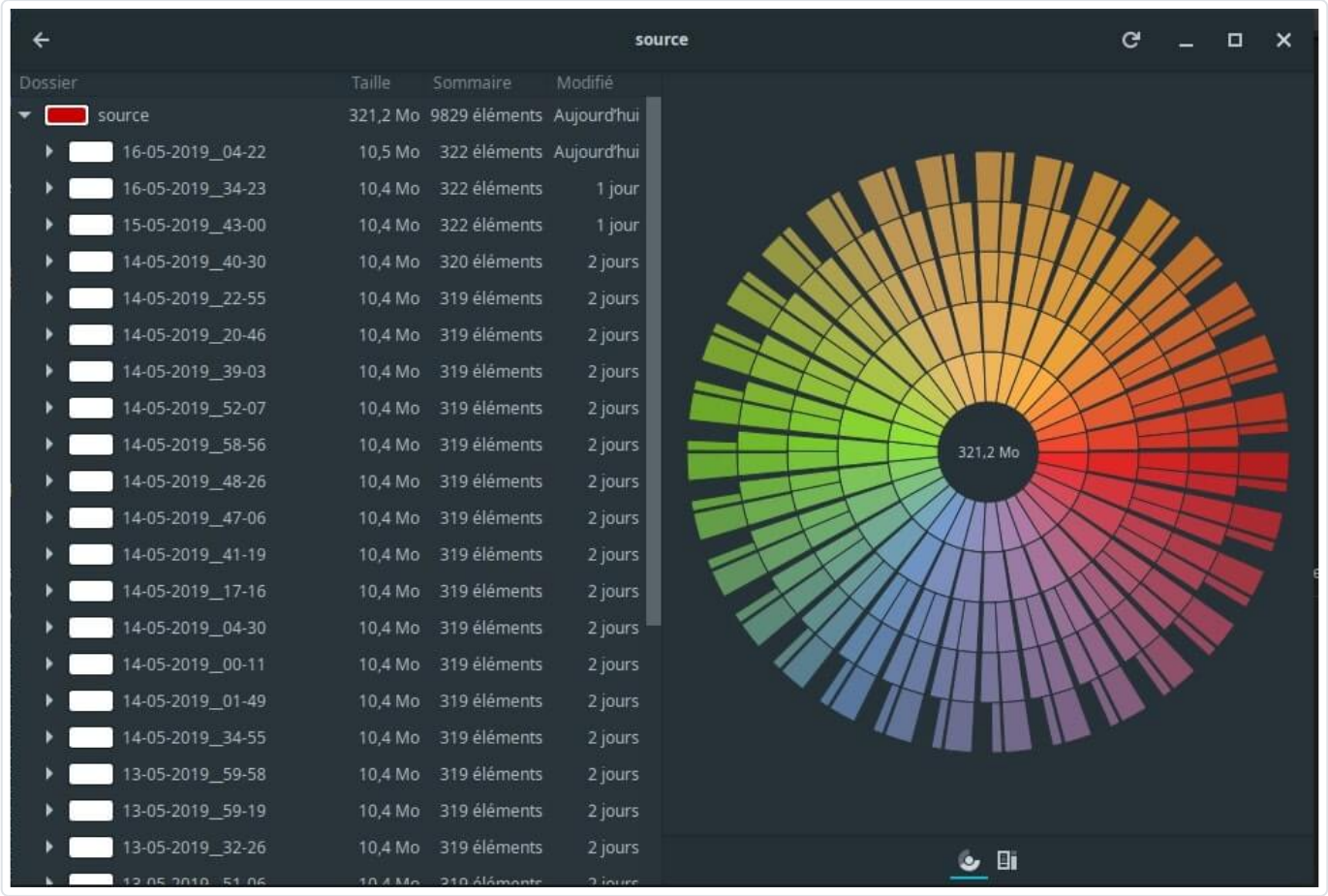
- **Backup/sources :**



Cette étape reste très importante même si vous utilisez git, car il peut arriver de déployer votre code sans forcement commiter (pour info git n'est pas un système de sauvegarde).

Voilà en cas de problème sur votre déploiement, vous pouvez facilement lancer une procédure de retour en arrière.

De plus les fichiers sources que vous sauvegardez pèsent vraiment pas grande chose en voici la preuve pour 32 déploiements (soit 32 sauvegardes)



Le déploiement

Enfin on y est arrivé à la partie déploiement !

Comme expliqué plus haut j'ai choisi d'ignorer un certain nombre de dossiers et de fichiers pour des raisons déjà citées plus haut et donc voici à quoi va ressembler le code source de mon action de déploiement :

```
deploy: backup ## Déploie les modifications vers le serveur de production
```

```
@echo -e "\n$(WARN_COLOR)- Déploiement des sources sur le serveur de $(ERROR_COLOR)p
@rsync -av \
--exclude '/var' \
--exclude '/.git' \
--exclude '.gitignore' \
--exclude '/.env' \
--exclude '/Makefile' \
--exclude '/src/DataFixtures' \
--exclude '/public/images/profiles' \
--delete \
. $(remote_ssh):$(remote_src)/
   @$(MAKE) -s clear_cache
```

Pour mon tout premier déploiement (quand je n'avais encore rien en prod) j'ai commenté la ligne suivante :

```
--exclude 'public/images/profiles' \
```

Pour que le dossier **public/images/profiles** soit créé automatiquement sur ma machine de production.

> **Attention**
>
> Il est important de modifier d'abord vos sources en local et de les envoyer ensuite en prod, car si vous faites l'inverse alors vous perdrez vos modifications à cause de l'option **`--delete`** qui supprimera automatiquement vos modifications sur le serveur de prod après le phase de déploiement

Ce n'est pas encore fini car il faut traiter la ligne suivante :

```
@$(MAKE) -s clear_cache
```

Cette ligne de code lancera l'action **`clear_cache`** qui permet de vider le cache du serveur distant pour que mes nouvelles modifications soient bien prises en compte.

Comme cette action n'existe pas alors il va falloir la créer et voici donc à quoi elle ressemblera :

```
clear_cache: ## Vide le cache du serveur de Prod
    @echo -e "\n$(WARN_COLOR)- Effacement du cache sur le serveur de $(ERROR_COLOR)pr
    @ssh $(remote_ssh) "php7.1-cli ~/devopssec/bin/console cache:clear"
```

Makefile final

Voila à quoi va ressembler le Makefile final :

```
.PHONY: deploy backup mediaBackup

.DEFAULT_GOAL= help

remote_ssh:=[UTILISATEUR]@[HOST]
remote_src:=[SOURCE_DISTANTE]

backup_location:=[DOSSIER_LOCAL_DE_BACKUP)


# Couleurs pour rendre jolie l'affichage
COM_COLOR   = \033[0;34m
OBJ_COLOR   = \033[0;36m
OK_COLOR    = \033[0;32m
ERROR_COLOR = \033[0;31m
WARN_COLOR  = \033[0;33m
NO_COLOR    = \033[m


backup_media:=$(backup_location)/media/

mediaBackup: ## Sauvegarde des fichiers lourds (images et PDF) en incrémental dans un
    @echo -e "\n$(WARN_COLOR)- Récupération des $(WARN_COLOR)medias$(WARN_COLOR) du s
    @rsync -auv \
            --delete \
        $(remote_ssh):$(remote_src)/public/images $(backup_media)
    @rsync -auv \
            --delete \
            $(remote_ssh):$(remote_src)/public/documents $(backup_media)


date_backup=$(shell date +'%d-%m-%Y__%H-%M-%S')
backup_source:=$(backup_location)/source/$(date_backup)/
```

```
backup: mediaBackup ## Crée un nouveau dossier et sauvegarde dedans les sources
    @echo -e "\n$(WARN_COLOR)- Récupération des $(WARN_COLOR)sources$(WARN_COLOR) du
    @rsync -auv \
    --exclude '/vendor' \
    --exclude '/var/cache' \
    --exclude '/public/media' \
    --exclude '/public/images' \
    --exclude '/public/documents' \
    $(remote_ssh):$(remote_src) $(backup_source)


deploy: backup ## Déploie les modifications vers le serveur de production
 @echo -e "\n$(WARN_COLOR)- Déploiement des sources sur le serveur de $(ERROR_COLOR)p
 @rsync -av \
 --exclude '/var' \
 --exclude '/.git' \
 --exclude '.gitignore' \
 --exclude '/.env' \
 --exclude '/Makefile' \
 --exclude '/src/DataFixtures' \
 --exclude '/public/images/profiles' \
 --delete \
 . $(remote_ssh):$(remote_src)/
    @$(MAKE) -s clear_cache

clear_cache: ## Vide le cache du serveur de Prod
    @echo -e "\n$(WARN_COLOR)- Effacement du cache sur le serveur de $(ERROR_COLOR)pr
    @ssh $(remote_ssh) "php7.1-cli ~/devopssec/bin/console cache:clear"

help: ## Affiche la description de chaque actions
 @grep -E '(^[a-zA-Z_-]+:.*?##.*$$)|(^##)' $(MAKEFILE_LIST) | awk 'BEGIN {FS = ":.*?#
```

Conclusion

J'ai essayé de trouver le meilleur compromis entre déploiement rapide et sécurité en ne sauvegardant que le strict minimum pour un possible retour en arrière.

Il y a bien sûr toujours moyen d'améliorer ce Makefile. D'ailleurs mon Makefile actuel est un peu plus fourni que celui-là car il s'adapte directement à mon type d'application. Par exemple j'ai une action qui me permet d'activer le mode maintenance automatiquement sur mon application en production (je ferrai un jour un tutoriel là-dessus car le système doit être au préalable être créé sur symfony lui-même). Cette option me facilite beaucoup la vie et ça me permet de réagir très

vite si jamais je découvre une faille ou un bug critique.

C'est un processus qui peut très bien s'intégrer lors d'une **intégration continue** et de **livraison continue**.

Si avez des suggestions d'améliorations ou des interrogations concernant le Makefile actuel alors n'hésitez pas à poster un commentaire, j'y répondrai avec grand plaisir.