Bachelor CDWFS
Versionning
2025-2026

Sauvegardes et restauration

Partie 1 : Sauvegardes

Les données

Les données qui doivent être sauvegardées sont la base de données ainsi que les éventuels fichiers générés (bon de commande, facture, etc.)
Le code, quant à lui, doit être versionné sur un dépôt.

Les risques

- Panne matérielle, Sinistres physiques
- Erreur humaine, Bugs logiciels / mises à jour ratées
- Cyberattaques

Sans sauvegarde :

- impossibilité de restaurer les données, arrêt de l'activité, pertes financières,
- responsabilité légale éventuelle,
- perte de confiance des utilisateurs ou clients.

Sauvegardes locales : outillage de base

Mesurer le volume de données

Commandes shell

```
sudo du -h --max-depth=1 cheminvers/le/dossier | sort -h
```

Commandes mysql

```
SELECT
  table_schema AS database_name,
  ROUND(SUM(data_length + index_length) / 1024 / 1024, 2) AS size_mb
FROM information_schema.tables
WHERE table_schema = 'nom_de_la_base';
```

Commande postgreSQL

```
SELECT pg_size_pretty(pg_database_size('nom_de_la_base'));
```

Quick Backup

https://makina-corpus.com/symfony/symfony-dbtoolsbundle-sauvegarder-restaurer-anonymiser-base-donn%C3%A9es

Minimum vital

Script de backup (db + fichier) avec rotation + cron

Exemple de script

```
#!/bin/bash

set -e  # arrêt immédiat en cas d’erreur

# ======================================
# VARIABLES

APP_NAME="symfony_app"
APP_DIR="/var/www/symfony-app"
UPLOADS_DIR="$APP_DIR/public/uploads"
BACKUP_ROOT="/mnt/backup"
BACKUP_DIR="$BACKUP_ROOT/$APP_NAME"
DB_HOST="localhost"
DB_NAME="symfony_app"
DB_USER="symfony_user"
DATE=$(date +"%Y-%m-%d_%H-%M")
RETENTION_DAYS=7
LOGFILE="/var/log/backup_${APP_NAME}.log"

# ======================================
# LOG

echo "Début sauvegarde : $DATE" >> "$LOGFILE"

# ======================================
# VÉRIFICATIONS

for cmd in rsync tar mysqldump gzip; do
  command -v "$cmd" >/dev/null 2>&1 || {
```

```
    echo "Commande manquante : $cmd" >> "$LOGFILE"
    exit 1
  }
done

[ -d "$BACKUP_ROOT" ] || {
  echo "Disque de sauvegarde non monté" >> "$LOGFILE"
  exit 1
}

# =====================================
# CRÉATION DU DOSSIER DE SAUVEGARDE

TODAY_BACKUP="$BACKUP_DIR/$DATE"
mkdir -p "$TODAY_BACKUP"

# =====================================
# SAUVEGARDE DES UPLOADS (archive)

echo "Sauvegarde des uploads..." >> "$LOGFILE"

tar czf "$TODAY_BACKUP/uploads.tar.gz" \
  -C "$UPLOADS_DIR" .

# =====================================
# SAUVEGARDE DE LA BASE DE DONNÉES

echo "Sauvegarde de la base de données..." >> "$LOGFILE"

mysqldump -h "$DB_HOST" -u "$DB_USER" -p "$DB_NAME" \
  | gzip > "$TODAY_BACKUP/db.sql.gz"

# Variante postgreSQL
pg_dump -h $DB_HOST -U $DB_USER $DB_NAME | gzip > db.sql.gz

# =====================================
# ROTATION DES SAUVEGARDES

echo "Rotation des sauvegardes (> $RETENTION_DAYS jours)..." >> "$LOGFILE"

find "$BACKUP_DIR" -mindepth 1 -maxdepth 1 -type d \
  -mtime +$RETENTION_DAYS -exec rm -rf {} \;

# =====================================
# FIN

echo "Sauvegarde terminée avec succès" >> "$LOGFILE"
```

Exemple de Cron (tous les jours à 2h)

Modifier les permissions sur le fichier

```
chmod +x /usr/local/bin/mon-script-backup.sh
```

Ouvrir la Crontab

```
crontab -e
```

Insérer un cron

```
0 2 * * * /usr/local/bin/backup_symfony.sh
```

Vers des sauvegardes vraiment pro

Types de sauvegardes

Sauvegarde complète

Inconvénient : espace de stockage
Avantage : simplicité et rapidité de restauration.

Sauvegarde différentielle

Chaque sauvegarde différentielle contient toutes les modifications apportées depuis la dernière sauvegarde complète. Pour restaurer à partir d'une sauvegarde différentielle, vous n'avez généralement besoin que de deux sauvegardes. Vous avez besoin de la dernière sauvegarde complète et de la dernière sauvegarde différentielle.
Inconvénient : processus de restauration plus complexe
Avantage : gain de stockage

Sauvegarde incrémentielle

Les sauvegardes incrémentales ne sauvegardent que les modifications apportées depuis la dernière sauvegarde. Elles ignorent tout ce qui a déjà été sauvegardé.
Inconvénient : processus de restauration plus complexe et plus risqué
Avantage : gain de stockage optimal. Possibilité d’augmenter la fréquence des sauvegardes.

Sauvegarde en temps réel

Les sauvegardes en temps réel minimisent les risques de perte de données. Les sauvegardes sont effectuées instantanément, dès que des changements se produisent.
Inconvénient : mise en place complexe.
Avantage : minimise le risque de perte des données.

Rotation des sauvegardes

La rotation de sauvegarde permet de conserver plusieurs versions de données dans le temps, sans explosion du volume de stockage. Exemple :

Fréquence	Conservation
Quotidienne	7 jours
Hebdomadaire	4–5 semaines
Mensuelle	6–12 mois

Réplication des sauvegardes

La stratégie 3-2-1 :

- 3 copies de vos données.
- 2 types de stockage (local et cloud).
- 1 copie hors site (exemple : S3).

Automatisation des sauvegardes

Quelle est la quantité de perte de données que vous considérez comme « acceptable » en cas de panne système ou d'autre incident de perte de données. Moins vous êtes prêt à perdre de données, plus vos sauvegardes seront fréquentes.

Sauvegardes à chaud

Une sauvegarde à chaud permet de sauvegarder un système en fonctionnement. Contrairement à une sauvegarde « à froid », aucun arrêt de service n'est nécessaire. Le risque principal : sauvegarder des données pendant qu'elles sont en cours de modification. Le défi consiste à maîtriser la cohérence des données.

Sauvegardes Cloud

Protocoles

SFTP

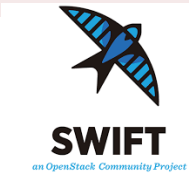
Simple d'utilisation. Inconvénient : ne gère pas les sauvegardes différentielles.

https://www.infomaniak.com/fr/support/faq/2545/creer-un-emplacement-sftp-sur-swiss-backup
https://www.infomaniak.com/fr/support/faq/2547/sauvegarder-des-donnees-avec-filezilla-sur-swiss-backup-sftp

Amazon S3 (Simple Storage Service)

Développé par Amazon Web Services, S3 est devenu un standard pour le stockage objet. Il permet de stocker et récupérer des données via une API REST.
https://aws.amazon.com/fr/s3/
https://skillbuilder.aws/learn/R54NZHEX5K/introduction-to-amazon-simple-storage-service-s3-franais/FJCXEYBM2D

Openstack Swift



Swift est un système de stockage open source. Il dispose également d'une API pour gérer les données.
Avantage : Peut être installé sur un serveur
Inconvénient : Courbe d'apprentissage forte.
https://docs.openstack.org/swift/latest/
https://wiki.openstack.org/wiki/Swift

Fournisseurs d'espace de stockage

Les données sont généralement organisées en "buckets" (conteneurs) contenant des objets identifiés par des clés uniques. Ces conteneurs sont ensuite accessible directement par des applications comme rclone installé sur votre serveur web.

Swiss Backup

Modalités : Swift, S3 ou SFTP
Compatible avec rClone, Duplicati...
Avantage : Données protégées, sauvegardes répliquées
https://www.infomaniak.com/fr/swiss-backup/stockage-cloud

Ovh Object Storage

Modalités : Swift ou S3
https://help.ovhcloud.com/csm/fr-public-cloud-storage-pcs-create-container?id=kb_article_view&sysparm_article=KB0047130

Autres Services

Scaleway

Fournisseur cloud français
Modalités : Compatible S3
Compatible avec rClone
Doc : https://www.scaleway.com/fr/object-storage/
Tuto : https://forestier.re/posts/2025-05-04-gerer-ses-backups-avec-rclone-et-scaleway-glacier/

Outscale Object Storage

Fournisseur de cloud computing français, certifié SecNumCloud.
https://fr.outscale.com/oos/
https://docs.outscale.com/fr/userguide/OUTSCALE-Object-Storage-OOS.html

Wasabi

Economiques, pas de frais de récupération.
https://wasabi.com/fr/direct/free-your-data

Digital Ocean

https://www.digitalocean.com/products/spaces

Cloudflare R2

https://www.cloudflare.com/lp/pg-r2-comparison-2/

Softwares

Rsync

« Simple outil de synchronisation »

Utilisable en ligne de commande. Sauvegarde incrémentielle : Il copie des fichiers d'un endroit à un autre en ne transférant que les différences (deltas) entre les fichiers source et destination, ce qui économise énormément de bande passante. Inconvénient : Les sauvegardes ne sont pas versionnées.
Tuto : https://blog.stephane-robert.info/docs/admin-serveurs/linux/rsync/

Rclone

« Rclone, c’est le rsync du stockage cloud ».

Comme Rsync, il permet de manipuler des fichiers/dossiers sur différents serveurs distants. Il est compatible avec les fournisseur d’espace de stockage cloud (Scaleway, Ovh, Swiss backup…). Il peut gérer des sauvegardes à chaud (checksums). Inconvénient : Les sauvegardes ne sont pas versionnées.
Tuto :

https://blog.stephane-robert.info/docs/cloud/outils/rclone/
https://forestier.re/posts/2025-05-04-gerer-ses-backups-avec-rclone-et-scaleway-glacier/

Restic

« Le spécialiste de la sauvegarde »

Comme Rclone, il est compatible avec les Cloud Provider (S3, Swift, …). Créé en 2014 en Go. Reconnu comme l'un des meilleurs outils de sauvegarde modernes.
Avantages :

- Snapshots de sauvegarde. Restauration granulaire à n'importe quelle date
- Gère la rétention et la rotation.
- Possibilité de sauvegarde à chaud
- Déduplication des données entre toutes les sauvegardes : Chaque sauvegarde crée un nouveau snapshot, mais seuls les segments nouveaux ou modifiés sont téléchargés. Les snapshots réutilisent les anciens segments autant que possible.

https://blog.stephane-robert.info/docs/cloud/outils/restic/
https://restic.net/

Autres applications

Duplicati

Sauvegardes complètes
Interface graphique et ligne de commande
https://domopi.eu/duplicati-un-logiciel-de-sauvegarde-universel/
https://prev-docs.duplicati.com/en/latest/?ref=domopi.eu

Borg

Ligne de commande
Sauvegarde incrémentielle
Auto-hébergé
https://blog.flozz.fr/2023/10/02/presentation-de-borgbackup-lun-des-meilleurs-outils-de-sauvegarde-disponibles-sous-linux/

Exécuter des sauvegardes

Script bash

Exemple :

Pré-requis : disposer d’un Cloud Provider configuré

```
#!/bin/bash
set -e  # Arrêt en cas d'erreur

# Dump de la base de données
mysqldump -u user -p'password' ma_base > /tmp/backup.sql

# Sauvegarde avec restic
restic -r s3:s3.amazonaws.com/mon-bucket backup /var/www /tmp/backup.sql

# Nettoyage
rm /tmp/backup.sql

# Notification
echo "Sauvegarde terminée" | mail -s "Backup OK" admin@example.com
```

Avantages du bash :

- Natif sur tous les systèmes Linux/Unix : aucune dépendance à installer.
- Parfait pour orchestrer des outils : restic, rsync, mysqldump, etc.
- Idéal pour cron : les tâches cron exécutent naturellement des scripts shell
- Excellente gestion des commandes système : redirections, pipes, codes de retour

Cron

Pour programmer l'exécution périodique de votre script de sauvegarde avec cron, voici comment procéder :

1. Préparer votre script

D'abord, rendez votre script exécutable :

```
chmod +x /chemin/vers/mon-script-backup.sh
```

2. Éditer la crontab

Ouvrez l'éditeur de crontab :

```
crontab -e
```

3. Ajouter une ligne cron

La syntaxe d'une ligne cron est :

```
minute heure jour mois jour_semaine commande
```

Exemples courants :

```
# Tous les jours à 2h du matin
0 2 * * * /chemin/vers/mon-script-backup.sh

# Tous les jours à 3h30
30 3 * * * /chemin/vers/mon-script-backup.sh

# Toutes les 6 heures
0 */6 * * * /chemin/vers/mon-script-backup.sh

# Tous les lundis à 1h
0 1 * * 1 /chemin/vers/mon-script-backup.sh

# Tous les dimanches à 23h
0 23 * * 0 /chemin/vers/mon-script-backup.sh

# Toutes les heures
0 * * * * /chemin/vers/mon-script-backup.sh
```

4. Bonnes pratiques

Ajouter la gestion des logs :

```
0 2 * * * /chemin/vers/mon-script-backup.sh >> /var/log/backup.log 2>&1
```

Spécifier le PATH et les variables :

```
PATH=/usr/local/bin:/usr/bin:/bin
0 2 * * * /chemin/vers/mon-script-backup.sh
```

Recevoir les erreurs par email :

```
MAILTO=admin@example.com
0 2 * * * /chemin/vers/mon-script-backup.sh
```

5. Vérifier vos tâches cron

Pour lister vos tâches cron actives :

```
crontab -l
```

Exemple complet de crontab

```
# Variables d'environnement
SHELL=/bin/bash
PATH=/usr/local/bin:/usr/bin:/bin
MAILTO=admin@example.com

# Sauvegarde quotidienne à 2h du matin
0 2 * * * /opt/scripts/backup-daily.sh >> /var/log/backup-daily.log 2>&1

# Sauvegarde hebdomadaire complète le dimanche à 3h
0 3 * * 0 /opt/scripts/backup-weekly.sh >> /var/log/backup-weekly.log 2>&1
```

Astuces

- **Tester d'abord manuellement** votre script avant de le mettre en cron
- **Utiliser des chemins absolus** dans votre script et dans cron
- **Vérifier les permissions** : cron s'exécute avec l'utilisateur qui a créé la tâche
- **Consulter les logs système** :

Debian/Ubuntu

```
grep CRON /var/log/syslog
```

systemd

```
journalctl -u cron
```

Pour aller plus loin

https://github.com/tiredofit/docker-db-backup

https://ottomatik.io/

https://ottomatik.io/blog

Partie 2 : Restauration

Décision de restaurer

Questions

Existe-t-il des alternatives ?

Pertes de données induites par la restauration ?

La procédure de sauvegarde va-t-elle induire une rupture de service ?

Fiabilité des sauvegarde ?

Responsabilité

La décision de restaurer une sauvegarde ne doit pas être uniquement technique : elle est organisationnelle et stratégique. La décision revient au "responsable métier" en concertation avec le responsable technique.

Feuille de route

1. Identifier ce qui doit être restauré

Le code peut être récupéré via le dépôt Git.

Données persistantes

- Base de données
- Fichiers uploadés

Configuration

- `.env`, `.env.local`
- clés JWT, secrets, variables d'environnement

Contexte

- Version PHP, Extensions PHP

2. Choisir la bonne sauvegarde

Identifier le snapshot / point de restauration

Vérifier :

- la date de la sauvegarde,
- la cohérence (pas une sauvegarde partielle), l'absence de corruption.

3. Préparer l'environnement de restauration

Restaurer d'abord sur un environnement de test si possible.

Avant toute écriture en production :

- Mettre l'application en maintenance (message utilisateur ou arrêt du service)
- Vérifier que :
  - le serveur cible est sain,
  - l'espace disque est suffisant,
  - les accès (SSH, DB, cloud) sont disponibles.
  - La version de php est la bonne, que les extensions php sont actives, …

4. Restaurer les fichiers

Tirer le code applicatif depuis le dépôt git si nécessaire.
Faire une sauvegarde ultime des fichiers uploadés (par sécurité).
Vérifier les droits d'écriture sur les dossiers

Exemple de script bash avec Rclone

```
#!/bin/bash

set -e  # stoppe le script à la première erreur

# ================================
# VARIABLES

REMOTE="backup:symfony-app/2025-12-20"
TARGET="/var/www/symfony-app"
CODE_DIR="app"
UPLOADS_DIR="public/uploads"
LOGFILE="/var/log/restore_symfony_files.log"

# ================================
# 1. Vérifications préalables

command -v rclone >/dev/null 2>&1 || {
  echo "rclone non installé" | tee -a "$LOGFILE"
  exit 1
}

[ -d "$TARGET" ] || {
  echo "Dossier cible introuvable : $TARGET" | tee -a "$LOGFILE"
  exit 1
}

# ================================
# 2. Sauvegarde de sécurité locale
echo "Sauvegarde locale de sécurité..." | tee -a "$LOGFILE"
```

```
tar czf "/tmp/symfony_app_before_restore_$(date +%F_%H%M).tar.gz" \
  "$TARGET/$UPLOADS_DIR"

# ================================
# 3. Restauration des fichiers

echo "Restauration des fichiers uploadés..." | tee -a "$LOGFILE"
rclone sync "$REMOTE/$UPLOADS_DIR" "$TARGET/$UPLOADS_DIR" \
  --progress --log-file="$LOGFILE"

# ================================
# 4. Permissions

echo "Correction des permissions..." | tee -a "$LOGFILE"

chown -R www-data:www-data "$TARGET"
chmod -R 755 "$TARGET"

# ================================
# FIN

echo "Restauration des fichiers terminée avec succès" | tee -a "$LOGFILE"
```

5. Restaurer la base de données

1. Sauvegarder l’état actuel (par sécurité).
2. Supprimer ou vider la base cible.
3. Restaurer le dump depuis la sauvegarde.

Exemple de script bash

```
#!/bin/bash

set -e  # stoppe à la première erreur

# ================================
# VARIABLES

REMOTE="backup:symfony-app/2025-12-20/db"
DUMP_FILE="symfony_db.sql.gz"
DB_HOST="localhost"
DB_NAME="symfony_app"
DB_USER="symfony_user"
LOGFILE="/var/log/restore_symfony_db.log"
```

```
# ================================
# 2. Sauvegarde de sécurité (DB actuelle)

echo "Sauvegarde de sécurité de la base actuelle..." | tee -a "$LOGFILE"

mysqldump -h "$DB_HOST" -u "$DB_USER" -p "$DB_NAME" \
  | gzip > "/tmp/${DB_NAME}_before_restore_$(date +%F_%H%M).sql.gz"

# ================================
# 3. Récupération du dump depuis le Cloud

echo "Téléchargement du dump depuis le Cloud..." | tee -a "$LOGFILE"

rclone copy "$REMOTE/$DUMP_FILE" /tmp \
  --progress --log-file="$LOGFILE"

# ================================
# 4. Restauration

echo "Restauration de la base de données..." | tee -a "$LOGFILE"

gunzip -c "/tmp/$DUMP_FILE" \
  | mysql -h "$DB_HOST" -u "$DB_USER" -p "$DB_NAME"
```

4. Vérifier :
   * encodage,
   * utilisateurs,
   * contraintes,
   * tables attendues.

6. Réadapter la configuration

Vérifier les variables :

- `DATABASE_URL`, clés d'API, secrets (`APP_SECRET`, JWT, etc.)

Lancer :

- `composer install` si nécessaire, `php bin/console cache:clear`,

7. Vérifier et valider

Avant remise en production,

Tester : page d'accueil, connexion utilisateur, fonctionnalités critiques. Vérifier que les uploads sont accessibles.

Vérifier les logs (`var/log`).

8. Remettre en production

Désactiver la maintenance
Informer les utilisateurs
Documenter :

- ce qui a été restauré,
- la date,
- les données perdues,
- la cause de l'incident.

Préparer les situations d'urgence

En situation d'urgence : stress élevé, pression métier, décisions rapides, erreurs coûteuses.
Tout ce qui n'a pas été préparé, documenté et testé devient un risque.
Le jour J : on exécute, on ne réfléchit pas à la syntaxe. Bonnes pratiques : disposer de scripts versionnés.

Documentation

A tout moment, savoir :

- Quelles données sont sauvegardées ?
- À quelle fréquence ? Combien de temps sont-elles conservées ?
- Où sont-elles stockées ? Qui y a accès ?
- Quelle est (sont) la (les) procédure(s) de restauration ?

Pouvoir accéder rapidement à :

- accès SSH,
- accès Cloud (Rclone / Restic),
- mots de passe DB,
- clés de chiffrement,
- secrets applicatifs.

Tester les restauration sur un environnement de test

Régulièrement, et après chaque changements majeurs

Définir les éléments critiques à vérifier
Restaurer une base de données, des fichiers, une application complète.

Documenter : Améliorer la documentation existante, Consigner les difficultés rencontrés, Evaluer la durée d'une restauration.

Plan de reprise après sinistre : https://objectfirst.com/fr/guides/data-security/disaster-recovery-plan/