Fondamentaux de ZFS

Documentation rédigée par : GADONNAUD Ewen

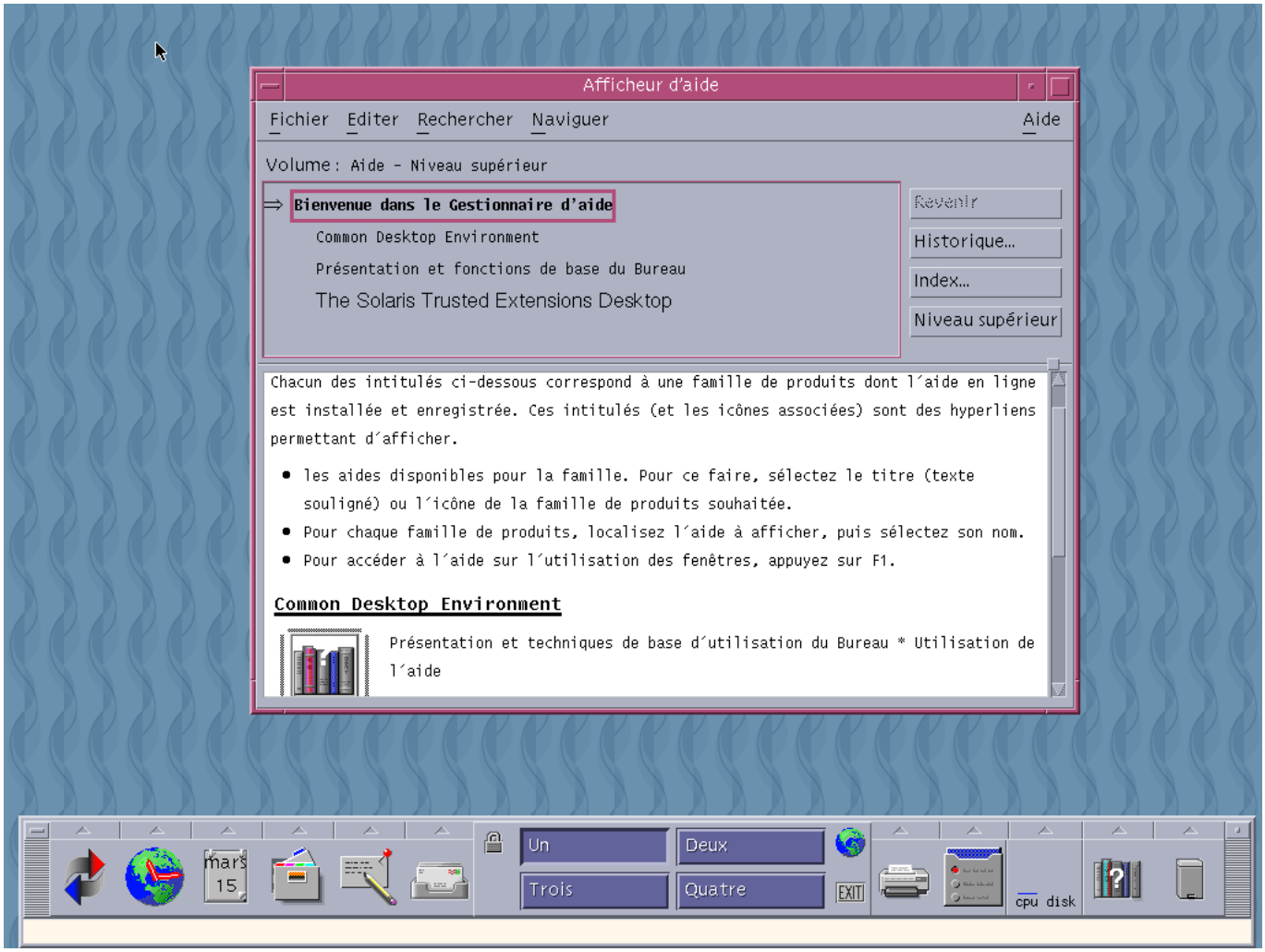
Formation : BTS SIO 1ère année - Option SISR

Établissement : Lycée Paul-Louis Courier, Tours

Date : Mars 2026

Environnement : Solaris 10 05/09 (Sun Microsystems) — VirtualBox 7.x — Windows 11

ZFS a été inventé par Sun Microsystems en 2006. Cette documentation est réalisée sur Solaris 10, le système d'exploitation original sur lequel ZFS a été conçu afin d'obtenir une expérience d'apprentissage en conditions authentiques.



1. Présentation de ZFS

ZFS (Zettabyte File System) est un système de fichiers développé par Sun Microsystems, introduit dans Solaris 10 en 2006. Il se distingue des systèmes de fichiers classiques (ext4, NTFS) par une approche radicalement différente : **ZFS fusionne le gestionnaire de volumes et le système de fichiers en une seule couche**.

Ce qui rend ZFS unique

Fonctionnalité	ZFS	ext4 / NTFS
Intégrité des données (checksums)	✅ Sur chaque bloc	❌ Inexistant
Snapshots natifs	✅ Instantanés, sans coût	❌ Inexistant nativement
RAID intégré (RAIDZ)	✅ Oui	❌ Non
Compression transparente	✅ Par dataset	LZNT1 (NTFS)
Rollback instantané	✅ Oui	❌ Impossible
Détection de corruption silencieuse	✅ Oui	❌ Non

Le principe fondamental : Copy-on-Write (CoW)

ZFS ne modifie **jamais** les données en place. Lors d'une écriture :

1. Les nouvelles données sont écrites dans un nouvel emplacement libre.
2. Les pointeurs sont mis à jour pour pointer vers les nouvelles données.
3. Les anciennes données restent intactes jusqu'à libération explicite.

C'est ce mécanisme qui rend les snapshots instantanés et gratuits : un snapshot, c'est simplement “geler les pointeurs actuels”.

---

2. Architecture : la hiérarchie ZFS

```
Disques physiques
        ↓
    zpool  (ex: monpool)
        ↓
    vdevs  (mirror, raidz, stripe...)
        ↓
  datasets  (monpool/data, monpool/backups...)
        ↓
snapshots / clones
```

Le Pool (zpool)

Le pool est la couche de base, il regroupe les disques physiques. Plusieurs modes de redondance sont disponibles :

Mode	Description	Disques minimum	Espace utile
stripe	Pas de redondance, perf maximale	1	100%
mirror	Copie identique sur 2 disques	2	50%
RAIDZ1	Équivalent RAID-5, tolère 1 panne	3	(n-1) × taille
RAIDZ2	Équivalent RAID-6, tolère 2 pannes	4	(n-2) × taille

RAIDZ vs RAID-5 classique : RAIDZ résout le “write hole” du RAID-5, un problème de corruption silencieuse lors d'un crash pendant une écriture. Grâce au CoW, ce scénario n'existe pas dans ZFS.

Le Dataset

Un dataset vit dans un pool et possède ses propres propriétés (compression, quota, etc.). Les

propriétés sont héritées par les datasets enfants.

```
monpool/           ← pool racine
monpool/data       ← dataset
monpool/backups    ← dataset
monpool/data/logs  ← dataset enfant (hérite des propriétés de data)
```

---

3. Mise en pratique

Environnement

- **Système** : Solaris 10 05/09 sous VirtualBox
- **Pool** : 3 disques virtuels SATA de 10 Go chacun
- **Disques détectés** : c2t0d0, c2t1d0, c2t2d0

Sur Solaris, après ajout de disques à chaud, la commande **devfsadm** est nécessaire pour que le système les détecte.

Création du pool RAIDZ1

```
zpool create monpool raidz /dev/dsk/c2t0d0 /dev/dsk/c2t1d0 /dev/dsk/c2t2d0
```

Vérification :

```
zpool status monpool
zpool list
```

Résultat obtenu :

```
NAME      SIZE   USED  AVAIL  CAP  HEALTH  ALTROOT
monpool  29,8G  189K  29,7G   0%  ONLINE  -
```

- Les 3 disques de 10 Go donnent **~20 Go utiles** (1 disque de parité en RAIDZ1).
- Le pool est monté automatiquement sur **/monpool**.

Création de datasets

```
zfs create monpool/data
zfs create monpool/backups
zfs list
```

Résultat :

```
NAME                USED   AVAIL  REFER  MOUNTPOINT
monpool             185K   19,5G  25,3K  /monpool
monpool/backups      24K   19,5G  24,0K  /monpool/backups
monpool/data         24K   19,5G  24,0K  /monpool/data
```

Les deux datasets **partagent** les 19,5 Go disponibles — pas de quota fixe par défaut. L'espace est alloué dynamiquement.

Compression

ZFS supporte plusieurs algorithmes de compression transparente, activables par dataset :

Algorithme	Taux	CPU	Usage recommandé
lzjb	Moyen	Quasi nul	Données actives (Solaris 10)
gzip-6	Bon	Modéré	Usage général
gzip-9	Excellent	Élevé	Archives, backups, données froides
lz4	Bon	Quasi nul	Données actives (OpenZFS moderne)
zstd	Excellent	Faible	Recommandé (OpenZFS moderne)

Sur Solaris 10, les algorithmes disponibles sont : **on \| off \| lzjb \| gzip \| gzip-[1-9]**. lz4 et zstd sont apparus plus tard dans OpenZFS.

Configuration appliquée :

```
zfs set compression=lzjb monpool/data       # données actives → algo rapide
zfs set compression=gzip-9 monpool/backups  # backups → compression maximale

zfs get compression monpool/data monpool/backups
```

```
NAME              PROPERTY     VALUE   SOURCE
monpool/backups   compression  gzip-9  local
monpool/data      compression  lzjb    local
```

La valeur **local** indique que la propriété est définie directement sur le dataset. Un dataset enfant **hériterait** automatiquement de cette propriété.

Quotas

Les quotas permettent de limiter l'espace consommé par un dataset :

```
zfs set quota=5g monpool/data
zfs get quota monpool/data
```

4. Snapshots

Un snapshot est une photo instantanée de l'état d'un dataset à un instant T. Grâce au CoW, il est **créé instantanément** et **ne consomme aucun espace** au moment de sa création — il grandit seulement au fur et à mesure que les données originales sont modifiées.

Création et rollback

```
# Copie de fichiers test
cp /etc/passwd /monpool/data/
cp /etc/hosts /monpool/data/

# Snapshot de l'état actuel
zfs snapshot monpool/data@v1

# Suppression des fichiers
rm /monpool/data/passwd /monpool/data/hosts

# Rollback instantané vers le snapshot
zfs rollback monpool/data@v1

# Les fichiers sont de retour
ls /monpool/data/
hosts  passwd
```

Le rollback ne réécrit aucune donnée — il repositionne simplement les pointeurs vers l'état du snapshot. L'opération est instantanée quelle que soit la taille du dataset.

5. Clones

Un clone est un dataset **inscriptible** créé à partir d'un snapshot. Contrairement à une copie classique :

- Au moment de la création, le clone **partage les mêmes blocs physiques** que le snapshot — zéro espace supplémentaire consommé.
- Le CoW entre en jeu uniquement lors d'une écriture dans le clone.

Création d'un clone

```
zfs clone monpool/data@v1 monpool/data-clone
```

```
ls /monpool/data-clone/
hosts  passwd
```

Vérification de l'espace partagé

```
zfs list -o name,used,refer monpool/data@v1 monpool/data-clone
```

```
NAME                  USED   REFER
monpool/data@v1      21,3K  27,3K
monpool/data-clone   23,3K  28,0K
```

Les blocs de hosts et passwd sont **partagés** entre le snapshot et le clone — ils ne comptent pas deux fois sur le disque.

Cas d'usage en production : cloner un snapshot d'une base de données de 500 Go pour un environnement de test prend quelques secondes et quasiment aucun espace, tant que les données ne divergent pas.

---

6. Vérification d'intégrité : zpool scrub

Le scrub est le mécanisme de vérification d'intégrité de ZFS. Il lit **chaque bloc** du pool, recalcule les checksums et les compare. Sur un pool miroir ou RAIDZ, il peut **autocorriger** les erreurs en lisant la copie saine.

```
zpool scrub monpool
zpool status monpool
```

Résultat :

```
scrub completed avec 0 erreurs
```

Planification en production

```
crontab -e
# Scrub automatique tous les dimanches à 2h
0 2 * * 0 /usr/sbin/zpool scrub monpool
```

Sur ext4 ou NTFS, il n'existe aucun équivalent au scrub. La **corruption silencieuse** (bit rot, secteur défaillant, RAM qui flippe un bit) peut s'accumuler pendant des années sans être détectée. C'est précisément ce problème que ZFS a été conçu pour résoudre chez Sun Microsystems.

Récapitulatif des commandes

Commande	Description
`zpool create <pool> raidz /dev/dsk/…`	Créer un pool RAIDZ1
`zpool status`	État du pool et des disques
`zpool list`	Liste des pools et espace disponible
`zpool scrub <pool>`	Lancer une vérification d'intégrité
`zfs create <pool/dataset>`	Créer un dataset
`zfs list`	Lister les datasets
`zfs set <propriété>=<valeur> <dataset>`	Définir une propriété
`zfs get <propriété> <dataset>`	Lire une propriété
`zfs snapshot <dataset>@<nom>`	Créer un snapshot
`zfs rollback <dataset>@<nom>`	Rollback vers un snapshot
`zfs clone <dataset>@<snap> <destination>`	Créer un clone

Ressources

- OpenZFS Documentation — documentation de référence du fork open source
- nex7.blogspot.com — blog technique approfondi sur ZFS
- Blog de Jeff Bonwick (Wayback Machine) — blog de l'inventeur de ZFS chez Sun

---

— Ewen 2026/03/15