Optimisation ZFS Proxmox VE 9 : ARC, L2ARC, SLOG et tuning avancé

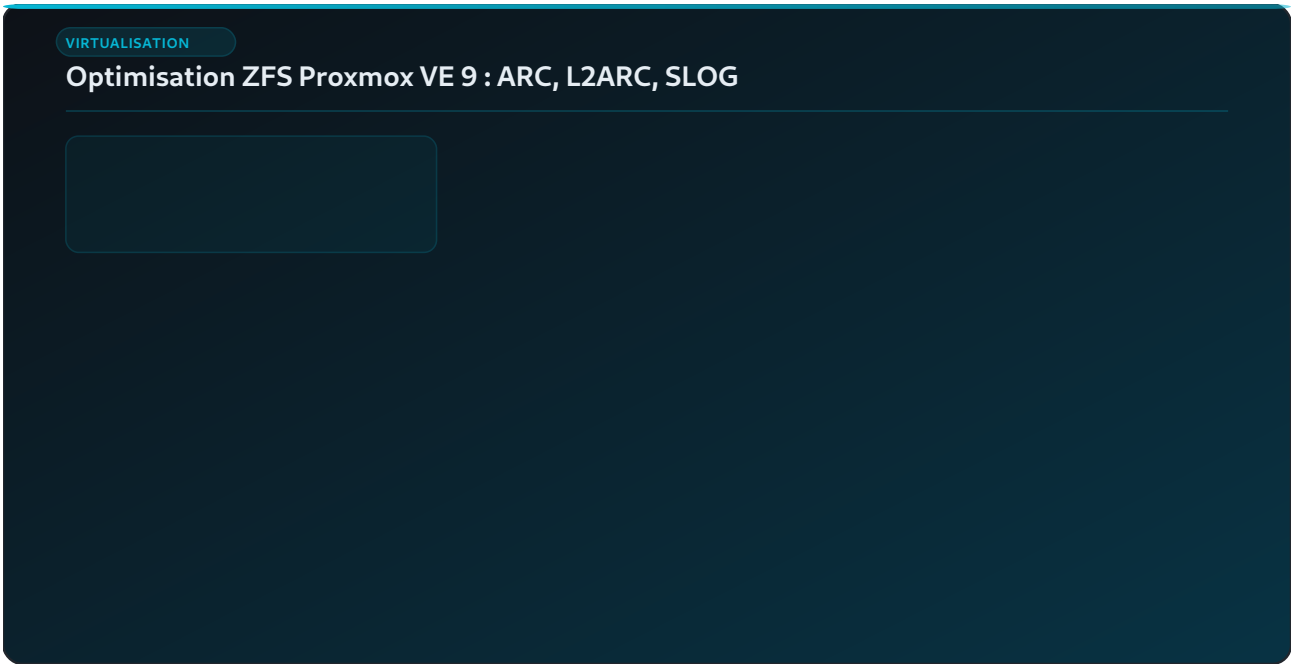
> Optimisation ZFS sur Proxmox VE 9 : dimensionner l'ARC (zfs_arc_max), ajouter L2ARC et SLOG NVMe, tuner compression lz4/zstd, recordsize et zvols de VM avec...

L'optimisation de ZFS sur Proxmox VE 9 est un exercice d'équilibre entre quatre ressources critiques : la RAM (cache ARC), le CPU (compression et checksums), les I/O (vdev, L2ARC, SLOG) et l'espace disque (snapshots, fragmentation). Proxmox VE 9 embarque OpenZFS 2.3.3 qui apporte deux évolutions majeures pour les administrateurs d'hyperviseurs : l'expansion à chaud des vdev RAIDZ-N permettant d'agrandir un pool RAIDZ sans reconstruction complète, et une refonte du comportement de l'ARC désormais plus agressif dans la consommation de RAM disponible. Cette seconde évolution rend le dimensionnement explicite de `zfs_arc_max` et `zfs_arc_min` indispensable sur tout hyperviseur hébergeant des VM, sous peine de subir des OOM-kill lors des pics de charge. Ce guide complet détaille l'ensemble des paramètres à maîtriser : dimensionnement de l'ARC, ajout de caches L2ARC et SLOG sur NVMe, réglage des datasets (recordsize, compression lz4/zstd, atime), tuning des zvols de VM, expansion RAIDZ à chaud, et maintenance préventive avec scrub et TRIM. Chaque section inclut des commandes de diagnostic pour mesurer l'impact avant et après chaque modification.

📋 EN BREF

- OpenZFS 2.3 sur Proxmox VE 9 : ARC plus agressif, expansion RAIDZ à chaud disponible
- Dimensionner explicitement `zfs_arc_max` et `zfs_arc_min` dans /etc/modprobe.d/zfs.conf
- Règle de base : ARC = 2 GiB + 1 GiB par TiB de stockage utile
- L2ARC utile uniquement si hit ratio ARC < 90 % sur pool HDD/SSD lents
- SLOG NVMe enterprise (avec PLP) pour VM hébergeant des bases de données

- Compression LZ4 ou ZSTD : quasi toujours gagnante, activer sur tous les pools
- Ne jamais dépasser 80 % d'occupation d'un pool ZFS en production



Proxmox VE 9, basé sur Debian 13 Trixie, embarque OpenZFS 2.3.3. Cette version apporte deux évolutions majeures pour les administrateurs de hyperviseurs. La première est l'expansion à chaud des vdev RAIDZ-N via `zpool attach` sur un RAIDZ existant, permettant d'agrandir un pool sans reconstruction complète. La seconde est la refonte du comportement de l'ARC : il peut désormais occuper jusqu'à environ 95 % de la RAM disponible avant de se rétracter, contre 50 % sous ZFS 2.2. La libération est censée être plus rapide, mais sur un hyperviseur où des VM démarrent simultanément ou subissent des pics de charge, ce comportement peut déclencher des OOM-kill si `zfs_arc_max` n'est pas explicitement borné.

ZFS est à la fois un système de fichiers, un gestionnaire de volumes et un système RAID logiciel. Sa robustesse repose sur trois mécanismes fondamentaux : checksums systématiques

(autoréparation), copy-on-write (snapshots quasi instantanés) et un cache mémoire adaptatif (l'ARC) qui est aussi sa principale source de tension avec les VM sur un hyperviseur Proxmox.

```
 # Vérifier la version OpenZFS et l'état général
pveversion -v | grep zfs    # Versions des paquets ZFS installés
zfs --version               # Version d'OpenZFS active (userland et module)
modinfo zfs | head -5       # Informations sur le module noyau ZFS chargé
cat /sys/module/zfs/version # Version du module ZFS effectivement actif
zpool status                # Santé des pools (erreurs, scrub, dégradation)
zpool list -v               # Capacité, fragmentation et arborescence vdev
zfs list -t all             # Tous datasets, zvols et snapshots avec leur taille
```

Pour une vue d'ensemble de l'architecture Proxmox avec ZFS, consultez notre guide Proxmox VE : hyperviseur KVM et LXC open source.

2. Le cache ARC : mécanisme et problématiques sur hyperviseur

L'ARC (Adaptive Replacement Cache) est un cache en RAM qui conserve à la fois les blocs de données récemment lus (MRU — Most Recently Used) et les blocs les plus fréquemment lus (MFU — Most Frequently Used). Contrairement au page cache classique de Linux, l'ARC est géré directement par ZFS et n'apparaît pas dans la mémoire « buffers/cache » de `free`, ce qui crée une confusion fréquente lors du diagnostic mémoire sur un hyperviseur Proxmox.

> ⚡ **Retour terrain**
>
> Pour une collectivité qui migrait de VMware vSphere vers Proxmox VE après la fin des licences perpétuelles, le point de friction principal était la migration des VMs avec des snapshots anciens et des disques fragmentés. J'ai développé un playbook de migration en 3 phases : consolidation des snapshots avec qemu-img, conversion OVA→qcow2, et

validation post-migration par comparaison MD5 des fichiers critiques. Sur 180 VMs, 94 % ont migré sans incident.

Sur un Proxmox VE 9 fraîchement installé avec ZFS comme système de fichiers racine, l'installateur écrit une limite ARC dans `/etc/modprobe.d/zfs.conf` : typiquement 10 % de la RAM physique, plafonnée à 16 GiB. Sans cette limite, l'ARC viserait par défaut environ 95 % de la RAM sous OpenZFS 2.3. Le problème typique sur un hyperviseur : l'ARC se remplit progressivement de blocs lus par les VM, puis lorsqu'une VM tente d'allouer de la RAM ou qu'un backup vzdump démarre, le système se retrouve en pression mémoire. ZFS doit libérer l'ARC, ce qui est plus coûteux qu'une libération de page cache standard, et peut momentanément bloquer les I/O ou provoquer un OOM-kill.

2.1 Statistiques et diagnostic de l'ARC

```
 # Diagnostic complet de l'ARC
arc_summary                # Vue d'ensemble : taille, hit ratio, MRU/MFU, prefetch
arc_summary -s arc         # Section spécifique : taille actuelle vs cible
arcstat 1 10               # Statistiques temps réel (10 mesures à 1 seconde)
cat /proc/spl/kstat/zfs/arcstats  # Compteurs bruts du noyau

# Taille actuelle de l'ARC en GiB
awk '/^size/{print $1, $3/1024/1024/1024" GiB"}' /proc/spl/kstat/zfs/arcstats

# Limites actuellement appliquées
cat /sys/module/zfs/parameters/zfs_arc_max  # Limite max (en octets)
cat /sys/module/zfs/parameters/zfs_arc_min  # Limite min (en octets)
free -h  # RAM globale (attention : ARC n'apparaît pas dans buff/cache)
```

Un hit ratio supérieur à 95 % indique un ARC bien dimensionné. Entre 80 % et 95 %, l'augmentation de la taille peut apporter un gain mesurable. En dessous de 80 %, soit l'ARC est sous-dimensionné, soit la charge est dominée par des écritures (l'ARC ne cache que les lectures et les métadonnées, pas les écritures).

3. Dimensionnement de l'ARC

Le dimensionnement de l'ARC sur un hyperviseur Proxmox doit équilibrer trois besoins : laisser suffisamment de RAM aux VM et conteneurs, garantir un cache de lecture efficace, et conserver une marge pour le système hôte et les pics de charge (backups vzdump, migrations live). La règle de pouce officielle Proxmox : **ARC = 2 GiB de base + 1 GiB par TiB de stockage utile**. Pour un pool de 8 TiB, on vise donc 10 GiB d'ARC. Sur un hyperviseur, prévoir en priorité : RAM hôte (4 à 8 GiB) + somme des RAM allouées aux VM (avec une marge de 20 %) + ARC. Si le total dépasse la RAM physique, réduire l'ARC en priorité avant de toucher aux VM.

RAM physique	ARC recommandé
16 GiB	2 GiB max — privilégier les VM
32 GiB	4 à 8 GiB selon volumétrie
64 GiB	8 à 16 GiB (limite par défaut PVE)
128 GiB	16 à 32 GiB selon nombre de VM
256 GiB et plus	32 à 64 GiB, rarement utile au-delà

3.1 Limitation immédiate à chaud (volatile)

```
 # Fixer zfs_arc_max à 8 GiB (immédiat, perdu au reboot)
echo $((8 * 1024 * 1024 * 1024)) > /sys/module/zfs/parameters/zfs_arc_max

# Si la nouvelle valeur de arc_max est <= arc_min, baisser d'abord arc_min
echo $((8 * 1024 * 1024 * 1024 - 1)) > /sys/module/zfs/parameters/zfs_arc_min
echo $((8 * 1024 * 1024 * 1024)) > /sys/module/zfs/parameters/zfs_arc_max

# Forcer la libération immédiate de l'ARC excédentaire
echo 3 > /proc/sys/vm/drop_caches
```

3.2 Limitation permanente (persistante au reboot)

```
 # Édition du fichier de configuration du module ZFS
cat > /etc/modprobe.d/zfs.conf << 'EOF'
# ARC max = 8 GiB, ARC min = 2 GiB
options zfs zfs_arc_max=8589934592
options zfs zfs_arc_min=2147483648
EOF

# Régénération obligatoire si root sur ZFS
update-initramfs -u -k all

# Redémarrer pour appliquer
reboot
```

Piège critique — zfs_arc_min : Si la valeur souhaitée de `zfs_arc_max` est inférieure ou égale à `zfs_arc_min` (qui vaut par défaut 1/32 de la RAM système), `zfs_arc_max` sera silencieusement ignoré. Sur un serveur disposant de 256 GiB de RAM, `zfs_arc_min` vaut 8 GiB par défaut — impossible de descendre l'ARC sous 8 GiB sans ajuster aussi `zfs_arc_min`. Toujours définir les deux paramètres explicitement dans `/etc/modprobe.d/zfs.conf`.

---

4. Caches secondaires : L2ARC et SLOG

Lorsque la RAM disponible pour l'ARC est insuffisante, deux mécanismes basés sur des SSD permettent d'étendre les capacités de mise en cache. Le L2ARC est un cache de lecture de second niveau ; le SLOG est un journal d'intentions d'écriture synchrone déporté. Ces dispositifs ne remplacent pas un ARC correctement dimensionné mais le complètent dans des scénarios spécifiques.

4.1 L2ARC : cache de lecture sur SSD

Le L2ARC reçoit les blocs évincés de l'ARC. Il sert essentiellement pour les workloads de lecture aléatoire sur de gros volumes de données froides. Sa gestion consomme elle-même de la

RAM (environ 70 octets par bloc indexé) : un L2ARC surdimensionné par rapport à l'ARC peut donc paradoxalement réduire les performances en saturant la RAM de métadonnées d'indexation.

```
 # Gestion du L2ARC
zpool add <pool> cache <device>       # Ajouter un SSD comme L2ARC
zpool remove <pool> <device>          # Retirer un L2ARC à chaud
zpool iostat -v <pool> 5              # Activité du L2ARC en temps réel
arc_summary -s l2arc                  # Statistiques L2ARC (hits, misses, taille)
echo 1 > /sys/module/zfs/parameters/l2arc_rebuild_enabled  # Persistance entre reboots
```

Le L2ARC n'apporte un gain que si : (1) le hit ratio ARC est inférieur à 90 %, (2) le pool sous-jacent est composé de HDD ou de SSD lents, (3) la charge est dominée par des lectures aléatoires. Sur un pool 100 % NVMe ou avec un ARC bien dimensionné, le L2ARC est généralement inutile voire contre-productif.

4.2 SLOG : journal d'écritures synchrones sur NVMe

Le ZIL (ZFS Intent Log) journalise les écritures synchrones (fsync, O_SYNC) pour garantir leur durabilité. Par défaut il réside dans le pool principal ; un SLOG dédié (SSD NVMe enterprise avec power-loss protection) déporte cette charge, réduit la latence des écritures synchrones et soulage le pool principal. Cas typiques d'utilisation : VM hébergeant des bases de données relationnelles (MariaDB, PostgreSQL, Oracle), NFS partagé avec synchronisation des écritures.

```
 # Gestion du SLOG
zpool add <pool> log <device>                  # Ajouter un SLOG (SSD enterprise PLP)
zpool add <pool> log mirror <dev1> <dev2>      # SLOG en miroir (recommandé en prod)
zpool remove <pool> <log-device>               # Retirer un SLOG à chaud
zfs get sync <dataset>                         # Politique de synchronisation
zfs set sync=standard <dataset>               # Défaut (respect des fsync applicatifs)
```

Attention : Ne jamais utiliser `sync=disabled` en production. Désactiver les écritures synchrones améliore les performances mais expose à la perte des dernières secondes d'écriture en cas de coupure brutale, y compris pour des fichiers déjà confirmés à l'application par fsync. Cela est inacceptable pour toute VM hébergeant des bases de données ou des transactions financières.

5. Optimisation des pools et datasets

Au-delà de l'ARC, plusieurs paramètres influencent fortement les performances et l'efficacité de stockage. Un mauvais réglage à la création d'un pool peut nécessiter une recréation complète — d'où l'importance de bien choisir ces paramètres dès l'installation ou la mise en service.

5.1 ashift : alignement physique (immuable)

Le paramètre `ashift` définit la taille minimale d'écriture du pool : `ashift=12` = 4 KiB (SSD/NVMe standards), `ashift=13` = 8 KiB (certains NVMe enterprise). Un ashift inférieur à la taille physique réelle provoque une amplification d'écriture massive et dégrade les performances sur le long terme. Sur Proxmox VE 9, l'installateur impose `ashift=12` par défaut, ce qui convient à la quasi-totalité des SSD/NVMe actuels.

```
 # Vérifier et configurer l'ashift
zpool get ashift <pool>                      # Vérifier (non modifiable après création)
zpool create -o ashift=12 <pool> <device>  # Créer avec ashift=12 (recommandé)
zpool create -o ashift=13 <pool> <device>  # Pour NVMe enterprise 8K
smartctl -a /dev/<dev> | grep -i 'sector size'  # Vérifier la taille de secteur physique
```

5.2 Compression : quasiment toujours gagnante

La compression LZ4 est extrêmement rapide (impact CPU négligeable même sur des serveurs chargés) et apporte un gain typique de 1,5x à 2x sur des données mixtes (OS, logs, données applicatives). ZSTD offre un meilleur ratio de compression au prix d'un peu plus de CPU, particulièrement intéressant pour les sauvegardes et archives. Sur PVE 9, ZSTD est activé par défaut sur les nouveaux pools créés via l'installateur. La compression s'applique uniquement aux nouvelles données écrites après activation, pas aux données existantes.

```
# Gestion de la compression
zfs get compression,compressratio <pool>   # Voir l'algo actif et le ratio obtenu
zfs set compression=lz4 <pool>             # LZ4 (équilibré, recommandé partout)
zfs set compression=zstd <pool>            # ZSTD (meilleur ratio, CPU modéré)
zfs set compression=zstd-3 <pool>          # ZSTD niveau 3 (compromis vitesse/ratio)
zfs set compression=off <pool>             # Désactiver (rarement justifié)
```

5.3 recordsize et volblocksize : taille de bloc logique

`recordsize` pour les datasets et `volblocksize` pour les zvols (disques de VM) déterminent la taille des blocs ZFS. Une valeur trop élevée gaspille l'ARC et provoque des lectures inutiles ; une valeur trop faible augmente la pression sur les métadonnées et fragmente le pool. Choisir selon le type de workload :

```
# Réglage des tailles de bloc
zfs get recordsize <dataset>               # Taille de bloc actuelle
zfs set recordsize=128K <dataset>          # Défaut, généraliste, fichiers mixtes
zfs set recordsize=16K <dataset>           # Bases de données (MariaDB, PostgreSQL)
zfs set recordsize=1M <dataset>            # Gros fichiers séquentiels (médias, backups)
zfs get volblocksize <pool>/vm-<id>-disk-0  # Taille de bloc d'un zvol VM
```

5.4 atime : désactiver pour réduire les écritures parasites

L'atime (access time) provoque une écriture à chaque lecture de fichier, ce qui est inutile dans 99 % des cas et amplifie l'usure des SSD. Le désactiver est une optimisation systématique recommandée sur tous les pools Proxmox.

```
# Gestion de l'atime
zfs get atime <pool>        # Vérifier l'état d'atime
zfs set atime=off <pool>    # Désactiver atime sur tout le pool (recommandé)
zfs set relatime=on <pool>  # Variante : met à jour atime sous certaines conditions
```

6. Optimisations spécifiques aux VM (zvols)

Les disques de VM sont stockés en zvols (volumes ZFS exposés en mode bloc). Leur réglage diffère des datasets classiques et a un impact direct sur les performances perçues dans la VM ainsi que sur l'occupation effective du pool. La `volblocksize` d'un zvol existant ne peut pas être modifiée après création — choisir la bonne valeur dès le départ est donc critique. Pour appliquer une nouvelle valeur à un zvol existant, il faut migrer le disque de la VM (move disk dans l'interface PVE) vers un storage configuré avec la nouvelle blocksize, ou recréer le disque puis restaurer le contenu.

6.1 volblocksize et amplification d'écriture en RAIDZ

En RAIDZ-1/2/3, chaque bloc écrit génère des blocs de parité. Avec `ashift=12` et `volblocksize=8K` (ancien défaut), un bloc de 8K en RAIDZ2 occupe 16K (8K + 4K + 4K de parité). Proxmox VE 8+ a relevé le défaut à 16K. Pour les RAIDZ, on recommande souvent 32K ou 64K pour réduire l'amplification.

```
 # Diagnostic et configuration des zvols
zfs get volblocksize,refreservation,used <pool>/vm-<id>-disk-0
pvesm set <storage> --blocksize 32K        # Modifier la blocksize par défaut du storage
cat /etc/pve/storage.cfg | grep -A 5 zfspool  # Vérifier la config storage
```

6.2 Cache ARC par zvol : primarycache et secondarycache

Pour des zvols hébergeant des bases de données qui disposent déjà de leur propre cache applicatif (innodb_buffer_pool, shared_buffers Postgres), il peut être judicieux de ne cacher que les métadonnées côté ARC. Cette configuration libère de la RAM pour d'autres usages tout en conservant les bénéfices du cache de métadonnées.

```
# Gestion du cache par zvol
zfs set primarycache=metadata <pool>/vm-<id>-disk-0  # ARC : metadata seulement
zfs set primarycache=all <pool>/vm-<id>-disk-0       # Comportement par défaut (tout)
zfs set secondarycache=metadata <pool>/vm-<id>-disk-0  # L2ARC : metadata seulement
zfs get primarycache,secondarycache <pool>/vm-<id>-disk-0  # Vérifier les politiques
```

6.3 Mode de cache disque côté Proxmox

Le paramètre `cache` du disque virtuel dans Proxmox interagit avec l'ARC. Le mode `none` (par défaut sur PVE 9) contourne le page cache de l'hôte mais profite tout de même de l'ARC ZFS. Le mode `writeback` active le page cache hôte en plus, ce qui peut doubler la consommation mémoire pour la même donnée — à éviter sur ZFS.

```
# Configuration du cache disque VM
qm config <VMID> | grep ^scsi                                  # Voir les options actuelles
qm set <VMID> --scsi0 <storage>:vm-<id>-disk-0,cache=none   # Mode none (recommandé ZFS)
qm set <VMID> --scsi0 <storage>:vm-<id>-disk-0,cache=writethrough  # Sûr mais lent
qm set <VMID> --scsi0 <storage>:vm-<id>-disk-0,discard=on,ssd=1    # TRIM + émul. SSD
```

Pour la configuration complète des paramètres de VM dans Proxmox, consultez notre guide d'optimisation Proxmox VE expert. Pour l'intégration ZFS avec la réplication entre nœuds, notre guide Proxmox réplication ZFS couvre les snapshots de réplication et la gestion des jobs `pvesr`.

---

7. Expansion RAIDZ à chaud avec OpenZFS 2.3

OpenZFS 2.3 introduit l'expansion à chaud des vdev RAIDZ-N, une fonctionnalité longtemps attendue de la communauté ZFS. Il est désormais possible d'ajouter des disques à un RAIDZ existant via `zpool attach`, sans reconstruction complète du pool et sans interruption de service. La résilverisation est progressive et s'effectue en arrière-plan, avec un impact limité sur les performances de production.

```
# Expansion RAIDZ à chaud (OpenZFS 2.3+)
zpool attach <pool> <raidz-vdev> <new-device>  # Ajouter un disque au RAIDZ
zpool status <pool>                            # Suivre la progression de l'expansion
zpool list -v                                  # Voir la nouvelle capacité disponible

# Vérifier que la feature est activée sur le pool
zpool get all <pool> | grep raidz_expansion
```

Cette fonctionnalité change fondamentalement la planification capacitaire des pools RAIDZ sur Proxmox. Pour les stratégies de sauvegarde associées aux pools ZFS, consultez notre guide de backup et restauration Proxmox.

---

8. Profils de configuration par type d'usage

Les paramètres ZFS optimaux varient selon le type de workload hébergé sur Proxmox. Voici les profils recommandés par cas d'usage, permettant de partir d'une base solide et d'ajuster ensuite selon les mesures.

8.1 Hyperviseur polyvalent (VM mixtes)

```
# Pool principal : compression LZ4, atime désactivé, recordsize par défaut
zfs set compression=lz4 <pool>
zfs set atime=off <pool>
# ARC : 8-16 GiB selon RAM disponible
# Zvols VM : volblocksize=16K (défaut PVE 8+), cache=none
# Pas de L2ARC sauf si pool HDD et hit ratio < 90 %
```

8.2 Serveur de bases de données

```
# Dataset dédié DB avec recordsize réduit
zfs set recordsize=16K <pool>/databases
zfs set compression=lz4 <pool>/databases
zfs set atime=off <pool>/databases
# SLOG NVMe enterprise en miroir pour les écritures synchrones
zpool add <pool> log mirror /dev/nvme0n1 /dev/nvme1n1
# Zvol VM DB : primarycache=metadata (innodb/pg gèrent leur cache)
zfs set primarycache=metadata <pool>/vm-<id>-disk-0
```

8.3 Stockage de sauvegardes (Proxmox Backup Server)

```
# Compression ZSTD pour meilleur ratio sur données compressibles
zfs set compression=zstd <pool>/backups
zfs set recordsize=1M <pool>/backups  # Gros fichiers séquentiels
zfs set atime=off <pool>/backups
# L2ARC inutile (lectures séquentielles, pas aléatoires)
# ARC : réduire (backups lisent rarement les mêmes blocs)
```

---

9. Maintenance préventive et supervision

La maintenance ZFS sur Proxmox est légère mais incontournable. Scrubs périodiques pour détecter la corruption silencieuse, TRIM régulier sur les SSD, gestion des snapshots pour éviter le remplissage du pool et supervision continue de l'occupation constituent les quatre piliers de la maintenance ZFS en production.

9.1 Scrub : vérification d'intégrité

Le scrub lit l'intégralité des données et vérifie chaque checksum. C'est la seule façon de détecter et corriger une corruption silencieuse (bit-rot), notamment sur des pools mixant SSD vieillissants et HDD. Proxmox planifie automatiquement un scrub mensuel via `/etc/cron.d/zfsutils-linux`.

```
# Gestion du scrub
zpool scrub <pool>              # Lancer un scrub manuel
zpool scrub -p <pool>           # Mettre le scrub en pause
zpool scrub -s <pool>           # Stopper un scrub en cours
zpool status <pool>             # Progression et erreurs détectées
cat /etc/cron.d/zfsutils-linux  # Planification automatique mensuelle
```

9.2 TRIM : maintenir les performances SSD

```
# Gestion du TRIM SSD
zpool trim <pool>               # TRIM manuel sur tous les SSD du pool
zpool status -t <pool>          # État du TRIM (en cours, dernier passage)
zpool get autotrim <pool>       # Vérifier si le TRIM automatique est actif
zpool set autotrim=on <pool>    # Activer le TRIM continu (recommandé SSD)
```

9.3 Gestion des snapshots et de l'occupation

Au-delà de 80 % d'occupation, les performances d'un pool ZFS chutent significativement (passage du mode best-fit au mode first-fit pour l'allocation des blocs). Les snapshots oubliés sont la cause la plus fréquente de saturation silencieuse : un snapshot retient toutes les données modifiées depuis sa création, y compris les fichiers apparemment supprimés.

```
# Supervision de l'occupation
zpool list -o name,size,alloc,free,capacity,frag
zfs list -t snapshot -o name,used,refer -s used  # Snapshots triés par espace consommé
zfs destroy <pool>/<dataset>@<snapshot>          # Supprimer un snapshot précis
zfs destroy -r <pool>/<dataset>@<snapshot>        # Supprimer snapshot et descendants
zfs list -t snapshot | wc -l                      # Compter le nombre total de snapshots
```

Pour la stratégie globale de sauvegarde avec Proxmox Backup Server et les snapshots ZFS, consultez notre guide des stratégies de backup Proxmox. Pour l'architecture réseau SDN qui interagit avec les performances de stockage des VM, voir notre guide SDN Proxmox 9. Pour la sécurisation de vos pools ZFS dans le cadre du durcissement global, consultez notre guide de durcissement Proxmox VE 9. La documentation officielle ZFS sur Proxmox est disponible sur le wiki Proxmox — ZFS on Linux.

10. Récapitulatif des commandes par domaine

Domaine	Commande	Usage
ARC	`arc_summary`	Diagnostic initial
ARC	`arcstat 1 10`	Monitoring temps réel
ARC	`cat /sys/module/zfs/parameters/zfs_arc_max`	Limite active
ARC	`echo $((8*1024**3)) > /sys/module/zfs/parameters/zfs_arc_max`	Réduire à chaud
Pool	`zpool status`	Santé des pools
Pool	`zpool iostat -v 5`	I/O temps réel par vdev
Pool	`zpool scrub <pool>`	Maintenance mensuelle
Dataset	`zfs set compression=lz4 <pool>`	Activer compression
Dataset	`zfs set atime=off <pool>`	Optimisation systématique
Zvol VM	`zfs get volblocksize <pool>/vm-*`	Audit amplification
VM	`qm config <VMID> | grep ^scsi`	Mode cache disque

FAQ sur l'optimisation ZFS dans Proxmox VE 9

Pourquoi zfs_arc_max semble-t-il ignoré sur Proxmox VE 9 ?

Le problème le plus courant est que `zfs_arc_max` est inférieur ou égal à `zfs_arc_min`. Par défaut, `zfs_arc_min` vaut 1/32 de la RAM physique — sur un serveur 256 GiB, c'est 8 GiB. Si vous tentez de fixer `zfs_arc_max` à 6 GiB, la valeur sera silencieusement ignorée. Toujours définir les deux paramètres explicitement dans `/etc/modprobe.d/zfs.conf` et régénérer l'initramfs avec `update-initramfs -u -k all`.

Quand faut-il ajouter un L2ARC à un pool Proxmox ?

Un L2ARC n'est utile que dans des conditions spécifiques : hit ratio ARC inférieur à 90 %, pool composé de HDD ou SSD lents, workload dominé par des lectures aléatoires. Sur un pool 100 % NVMe, l'ajout d'un L2ARC est contre-productif car les NVMe sont déjà plus rapides que le L2ARC, et sa gestion consomme de la RAM précieuse (~70 octets par bloc indexé). Mesurer avec `arcstat` et `arc_summary` avant tout investissement en SSD L2ARC.

Quelle compression choisir entre LZ4 et ZSTD pour les VM Proxmox ?

LZ4 est le choix par défaut pour les zvols de VM : sa décompression est extrêmement rapide (plusieurs GiB/s) et son impact CPU est négligeable même sous forte charge. ZSTD offre un meilleur ratio de compression (15 à 30 % de plus que LZ4 sur des données mixtes) au prix d'un CPU légèrement plus sollicité en écriture. Recommandation : LZ4 sur les zvols de VM en production intensive, ZSTD sur les datasets de sauvegardes et archives.

Peut-on modifier le recordsize d'un dataset ZFS existant ?

Oui, mais seulement pour les nouvelles données : modifier `recordsize` sur un dataset existant n'affecte pas les blocs déjà écrits. Pour appliquer la nouvelle valeur à l'ensemble du dataset, il faut réécrire les données (par exemple via un `zfs send | zfs receive` vers un nouveau dataset avec la bonne valeur). La `volblocksize` d'un zvol de VM, en revanche, est figée à la création et ne peut pas être modifiée autrement qu'en recréant le zvol.

Comment détecter une fragmentation excessive sur un pool ZFS Proxmox ?

La commande `zpool list -o name,size,alloc,free,capacity,frag` affiche le taux de fragmentation. Un taux supérieur à 50 % indique une dégradation des performances d'écriture séquentielle. La fragmentation est inévitable et progresse avec les snapshots et les suppressions ; elle se résorbe naturellement lors d'une résilverisation ou d'un export/import de pool. Maintenir l'occupation sous 80 % est la meilleure prévention.

L'expansion RAIDZ à chaud d'OpenZFS 2.3 est-elle stable sur Proxmox VE 9 ?

L'expansion RAIDZ à chaud (`zpool attach` sur un RAIDZ existant) est disponible dans OpenZFS 2.3 (Proxmox VE 9) et est considérée comme stable pour les RAIDZ-1 et RAIDZ-2. La résilverisation s'effectue en arrière-plan sans interruption de service. Cette fonctionnalité modifie la structure interne du pool de façon irréversible ; un snapshot de sauvegarde complet avant l'opération reste fortement recommandé. Consulter les notes officielles OpenZFS sur l'expansion RAIDZ pour les limitations connues.

À RETENIR

À retenir

OpenZFS 2.3 (Proxmox VE 9) rend l'ARC plus agressif : toujours définir `zfs_arc_max` ET `zfs_arc_min` dans `/etc/modprobe.d/zfs.conf` , puis régénérer l'initramfs

Règle de dimensionnement ARC : 2 GiB de base + 1 GiB par TiB de stockage utile, avec une réservation de 4-8 GiB pour l'hôte Proxmox

Activer la compression LZ4 ou ZSTD et désactiver atime sur tous les pools : gain de performance et d'espace sans coût notable sur le CPU

La `volblocksize` d'un zvol de VM est fixée à la création — choisir 16K à 32K selon l'usage, impossible de modifier après sans recréer le zvol

L2ARC : uniquement si hit ratio ARC inférieur à 90 % sur pool HDD/SSD lents ; inutile et contre-productif sur pool 100 % NVMe

SLOG NVMe enterprise avec power-loss protection pour VM hébergeant des bases de données — ne jamais utiliser `sync=disabled`

Surveiller l'occupation des pools : alerte à 80 %, dégradation sévère au-delà de 90 % — les snapshots oubliés sont la première cause de saturation silencieuse

> Scrub mensuel automatique (planifié par Proxmox via cron) + TRIM SSD actif (`autotrim=on`) = maintenance ZFS de base incontournable

---

Sources et références

[ANSSI — Recommandations de sécurité relatives aux hyperviseurs]

[NIST SP 800-125 — Guide to Security for Full Virtualization]
