Ubuntu 20.04 Root sur ZFS pour Raspberry Pi

Table of Contents

- Ubuntu 20.04 Root sur ZFS pour Raspberry Pi
  - Configuration requise
  - Chiffrement
  - Étape 1 : Formatage du disque
  - Étape 2 : Configurer ZFS
  - Étape 3 : Installation du système
    - Créer un ensemble de données de système de fichiers pour agir en tant que conteneur :
    - Créer un ensemble de données de système de fichiers pour le système de fichiers racine :
    - Créer des ensembles de données :
    - Facultatif : Ignorer les requêtes synchrones :
    - Copier le système dans les systèmes de fichiers ZFS :
  - Étape 4 : Configuration du système
    - Configurer le nom d'hôte :
    - Arrêter zed :
    - Chroot dans le système de fichier
    - Configurer un environnement système de base :
    - Facultatif : monter un tmpfs sur /tmp
    - Patcher une dépendance pour le chiffrement natif ZFS ou LUKS :
    - Correction de l'ordre de montage du système de fichiers :
    - Facultatif (mais fortement recommandé) : Faciliter le démarrage du débogage :
    - Redémarrer :
  - Étape 5 : premier démarrage
    - Supprimer la partition ext4 et développer la partition ZFS :
    - Créer un compte utilisateur :
    - Redémarrer :
    - Développer le pool ZFS :
    - Supprimer l'utilisateur ubuntu :
  - Étape 6 : Installation complète du logiciel
    - Facultatif : Supprimer cloud-init :
    - Facultatif : supprimer les autres packages de stockage :
    - Facultatif : installer un environnement GUI complet :
    - Facultatif (mais recommandé) : désactiver la compression des journaux :
    - Redémarrer :
  - Étape 7 : Nettoyage final
    - Facultatif : pour les installations LUKS uniquement, sauvegarder l'en-tête LUKS :

Ce HOWTO utilise une carte SD physique entière.
Sauvegarder les données. Toutes les données existantes seront perdues.

Configuration requise

- Un Raspberry Pi 4 B. (Si vous chercher à installer sur un PC ordinaire, voir Ubuntu 20.04 Root sur ZFS.)
- Serveur Ubuntu 20.04.2 (« Focal ») pour Raspberry Pi 4
- Une carte microSD. Pour des recommandations, voir la comparaison des performances de Jeff Geerling.
- Un système Ubuntu (avec la possibilité d'écrire sur la carte SD) autre que le Raspberry Pi cible.
- 4 Gio de mémoire sont recommandés. N'utiliser pas la déduplication, car elle nécessite d'énormes quantités de RAM. L'activation de la déduplication est un changement permanent qui ne peut pas être facilement annulé.
- Un Raspberry Pi 3 B/B+ fonctionnerait probablement (car le Pi 3 est en 64 bits, bien qu'il ait moins de RAM), mais n'a pas été testé. Veuiller signaler vos résultats (bons ou mauvais) en utilisant le lien ci-dessous.

Chiffrement

Le cryptage n'a pas encore été testé sur le Raspberry Pi.

Ce guide prend en charge trois options de cryptage différentes : non crypté, cryptage natif ZFS et LUKS. Avec n'importe quelle option, toutes les fonctionnalités ZFS sont entièrement disponibles.

- Unencrypted ne crypte rien, bien sûr. En l'absence de cryptage, cette option a naturellement les meilleures performances.
- Le ciffrement natif ZFS chiffre les données et la plupart des métadonnées dans le pool racine. Il ne crypte pas les noms ou propriétés d'ensembles de données ou d'instantanés. Le pool de démarrage n'est pas du tout chiffré, mais il ne contient que le chargeur de démarrage, le noyau et l'initrd. (À moins que vous ne mettier un mot de passe dans /etc/fstab, il est peu probable que l'initrd contienne des données sensibles.) Le système ne peut pas démarrer sans que la phrase secrète ne soit entrée sur la console. Les performances sont bonnes. Comme le cryptage se produit dans ZFS, même si plusieurs disques (topologies miroir ou raidz) sont utilisés, les données ne doivent être cryptées qu'une seule fois.
- LUKS crypte presque tout. Les seules données non cryptées sont le chargeur de démarrage, le noyau et l'initrd. Le système ne peut pas démarrer sans que la phrase secrète ne soit saisie sur la console. Les performances sont bonnes, mais LUKS se trouve sous ZFS, donc si plusieurs disques (topologies miroir ou raidz) sont utilisés, les données doivent être chiffrées une fois par disque.

Étape 1 : Formatage du disque

Les commandes de cette étape sont exécutées sur le système autre que le Raspberry Pi.

Ce guide vous propose un travail supplémentaire afin que la partition stock ext4 puisse être supprimée.

Télécharger et décompresser l'image officielle :

```
curl -O
https://cdimage.ubuntu.com/releases/20.04.2/release/ubuntu-20.04.2-preinstal
led-server-arm64+raspi.img.xz
xz -d ubuntu-20.04.2-preinstalled-server-arm64+raspi.img.xz

# ou combiner-les pour décompresser au fur et à mesure du téléchargement :
curl
https://cdimage.ubuntu.com/releases/20.04.2/release/ubuntu-20.04.2-preinstal
led-server-arm64+raspi.img.xz | \
    xz -d > ubuntu-20.04.2-preinstalled-server-arm64+raspi.img
```

Vider la table de partition pour l'image :

```
sfdisk -d ubuntu-20.04.2-preinstalled-server-arm64+raspi.img
```

Cela affichera ceci :

```
label: dos
label-id: 0x4ec8ea53
device: ubuntu-20.04.2-preinstalled-server-arm64+raspi.img
unit: sectors
<name>.img1 : start=        2048, size=      524288, type=c, bootable
<name>.img2 : start=      526336, size=     5839840, type=83
```

Les nombres importants sont 524288 et 5839840. Stocker-les dans des variables :

```
export BOOT=524288
export ROOT=5839840
```

Créer un script de partition :

```
vi partitions.sh
```

avec le contenu suivant :

```
cat << EOF
```

```
label: dos
unit: sectors

1 : start=  2048,  size=$BOOT, type=c, bootable
2 : start=$((2048+BOOT)),  size=$ROOT, type=83
3 : start=$((2048+BOOT+ROOT)), size=$ROOT, type=83
EOF
```

Connecter la carte SD :

Connecter la carte SD à une machine autre que le Raspberry Pi cible. Si des systèmes de fichiers sont montés automatiquement (par exemple par GNOME), démonter-les. Déterminer le nom du périphérique (qui est presque certainement comme indiqué ci-dessous) et définisser-le dans une variable :

```
DISK=/dev/mmcblk0
```

Effacer les anciennes étiquettes ZFS :

```
sudo zpool labelclear -f ${DISK}
```

Si une étiquette ZFS existe toujours à partir d'un système/d'une tentative précédente, l'extension du pool entraînera un système non amorçable.

si les utilitaires ZFS ne sont pas déjà installé , on peut les installer avec : `sudo apt install zfsutils-linux` Alternativement, on peut mettre à zéro toute la carte SD avec : `sudo dd if=/dev/zero of=${DISK} bs=1M status=progress`

Supprimer les partitions existantes :

```
echo "label: dos" | sudo sfdisk ${DISK}
sudo partprobe
ls ${DISK}*
```

S'assurer qu'il n'y a pas de partitions, juste le fichier pour le disque lui-même. Cette étape n'est pas strictement nécessaire ; il existe pour attraper les problèmes.

Créer les partitions :

```
sh -u partitions.sh | sudo sfdisk $DISK
```

Monter l'image en loop :

```
IMG=$(sudo losetup -fP --show \
```

```
          ubuntu-20.04.2-preinstalled-server-arm64+raspi.img)
```

Copier les données du chargeur de démarrage :

```
sudo dd if=${IMG}p1 of=${DISK}p1 bs=1M
```

Effacer les anciennes étiquettes de la partition 2 :

```
sudo wipefs -a ${DISK}p2
```

Si un système de fichiers avec l'étiquette inscriptible de l'image Ubuntu est toujours présent dans la partition 2, le système ne démarrera pas initialement.

Copier les données du système de fichiers racine :

```
# la destination est p3, pas p2.
sudo dd if=${IMG}p2 of=${DISK}p3 bs=1M status=progress conv=fsync
```

Démonter l'image :

```
sudo losetup -d $IMG
```

Démarrer le Raspberry Pi.

Placer la carte SD dans le Raspberry Pi. Démarrer et se connecter (par exemple via SSH) avec ubuntu comme nom d'utilisateur et mot de passe. Lorsqu'on utilise SSH, il faut un peu de temps à cloud-init pour activer les connexions par mot de passe au premier démarrage. Définir un nouveau mot de passe lorsque on y est invité et se connecter à nouveau en utilisant ce mot de passe. Si le SSH local est configuré pour utiliser ControlPersist, il faudra tuer le processus SSH existant avant de se connecter pour la deuxième fois.

Étape 2 : Configurer ZFS

Devenir root :

```
sudo -i
```

Définir une variable avec le nom du disque :

```
DISK=/dev/mmcblk0
```

Sur le Pi, c'est toujours mmcblk0.

Installer ZFS :

mise à jour appropriée

```
apt install pv zfs-initramfs
```

Étant donné qu'il s'agit du premier démarrage, on peut obtenir En attente du verrouillage du cache car les mises à niveau sans surveillance s'exécutent en arrière-plan. Attender qu'il se termine.

Créer le pool racine :

Choisisser l'une des options suivantes :

- Non crypté :

```
zpool create \
    -o ashift=12 \
    -O acltype=posixacl -O canmount=off -O compression=lz4 \
    -O dnodesize=auto -O normalization=formD -O relatime=on \
    -O xattr=sa -O mountpoint=/ -R /mnt \
    rpool ${DISK}p2
```

Le cryptage n'a pas encore été testé sur le Raspberry Pi.

- Cryptage natif ZFS :

```
zpool create \
    -o ashift=12 \
    -O encryption=aes-256-gcm \
    -O keylocation=prompt -O keyformat=passphrase \
    -O acltype=posixacl -O canmount=off -O compression=lz4 \
    -O dnodesize=auto -O normalization=formD -O relatime=on \
    -O xattr=sa -O mountpoint=/ -R /mnt \
    rpool ${DISK}p2
```

- LUC :

```
cryptsetup luksFormat -c aes-xts-plain64 -s 512 -h sha256 ${DISK}p2
cryptsetup luksOpen ${DISK}-part4 luks1
zpool create \
    -o ashift=12 \
    -O acltype=posixacl -O canmount=off -O compression=lz4 \
    -O dnodesize=auto -O normalization=formD -O relatime=on \
    -O xattr=sa -O mountpoint=/ -R /mnt \
```

```
rpool /dev/mapper/luks1
```

L'utilisation de `ashift=12` est recommandée ici car de nombreux disques ont aujourd'hui 4 KiB (ou plus) de secteurs physiques, même s'ils présentent 512 B de secteurs logiques. De plus, un futur disque de remplacement peut avoir 4 secteurs physiques de Ko (auquel cas `ashift=12` est souhaitable) ou des secteurs logiques de 4 Ko (auquel cas ashift=12 est requis).

La définition de `-O acltype=posixacl` active les ACL POSIX globalement. Si on ne le souhaite pas, supprimer cette option, mais ajouter plus tard `-o acltype=posixacl` (« o » minuscule) à la création de zfs pour /var/log, car journald nécessite des ACL. De plus, la désactivation des ACL interrompt apparemment la gestion d'umask avec NFSv4.

La définition de normalisation=formD élimine certains cas de coin liés à la normalisation des noms de fichiers UTF-8. Cela implique également utf8only=on, ce qui signifie que seuls les noms de fichiers UTF-8 sont autorisés. Si on souhaite prendre en charge les noms de fichiers non UTF-8, ne pas utiliser pas cette option.

`recordsize` n'est pas défini (en le laissant à la valeur par défaut de 128 Ko). Si on veut le régler (par exemple, `-o recordsize=1M`), consulter ces différents articles de blog.

Définir `relatime=on` est un juste milieu entre le comportement `atime` classique de POSIX (avec son impact significatif sur les performances) et `atime=off` (qui offre les meilleures performances en désactivant complètement les mises à jour atime). Depuis Linux 2.6.30, `relatime` est la valeur par défaut pour les autres systèmes de fichiers. Voir la documentation de RedHat pour plus d'informations.

La définition de `xattr=sa` améliore considérablement les performances des attributs étendus. Dans ZFS, les attributs étendus sont utilisés pour implémenter les ACL POSIX. Les attributs étendus peuvent également être utilisés par les applications de l'espace utilisateur. Ils sont utilisés par certaines applications GUI de bureau. Ils peuvent être utilisés par Samba pour stocker les ACL Windows et les attributs DOS ; ils sont requis pour un contrôleur de domaine Samba Active Directory. `xattr=sa` est spécifique à Linux. Si on déplace le pool `xattr=sa` vers une autre implémentation OpenZFS en plus de ZFS-on-Linux, les attributs étendus ne seront pas lisibles (bien que les données le soient). Si la portabilité des attributs étendus est importante, omettre le `-O xattr=sa` ci-dessus. Même si on ne veut pas de `xattr=sa` pour l'ensemble du pool, on peut probablement l'utiliser pour /var/log.

S'assurer d'inclure la partie `-part4` du chemin du lecteur. Omettre cela, spécifie le disque entier, que ZFS repartitionnera ensuite, et on perd la ou les partitions du chargeur de démarrage.

Le chiffrement natif ZFS est par défaut `aes-256-ccm`, mais la valeur par défaut a été modifiée en amont par `aes-256-gcm`. AES-GCM semble être généralement préféré à AES-CCM, est plus rapide maintenant et le sera encore plus à l'avenir.

Pour LUKS, la taille de clé choisie est de 512 bits. Cependant, le mode XTS nécessite deux clés, la clé LUKS est donc divisée en deux. Ainsi, `-s 512` signifie AES-256.

Le mot de passe sera probablement le maillon le plus faible. Il faut le choisir avec rigueur.

Étape 3 : Installation du système

Créer un ensemble de données de système de fichiers pour agir en tant que conteneur :

```
zfs create -o canmount=off -o mountpoint=none rpool/ROOT
```

Créer un ensemble de données de système de fichiers pour le système de fichiers racine :

```
UUID=$(dd if=/dev/urandom bs=1 count=100 2>/dev/null |
    tr -dc 'a-z0-9' | cut -c-6)

zfs create -o canmount=noauto -o mountpoint=/ \
    -o com.ubuntu.zsys:bootfs=yes \
    -o com.ubuntu.zsys:last-used=$(date +%s) rpool/ROOT/ubuntu_$UUID
zfs mount rpool/ROOT/ubuntu_$UUID
```

Avec ZFS, il n'est normalement pas nécessaire d'utiliser une commande de montage (soit mount, soit zfs mount). Cette situation est une exception en raison de canmount=noauto.

Créer des ensembles de données :

```
zfs create -o com.ubuntu.zsys:bootfs=no \
    rpool/ROOT/ubuntu_$UUID/srv
zfs create -o com.ubuntu.zsys:bootfs=no -o canmount=off \
    rpool/ROOT/ubuntu_$UUID/usr
zfs create rpool/ROOT/ubuntu_$UUID/usr/local
zfs create -o com.ubuntu.zsys:bootfs=no -o canmount=off \
    rpool/ROOT/ubuntu_$UUID/var
zfs create rpool/ROOT/ubuntu_$UUID/var/games
zfs create rpool/ROOT/ubuntu_$UUID/var/lib
zfs create rpool/ROOT/ubuntu_$UUID/var/lib/AccountsService
zfs create rpool/ROOT/ubuntu_$UUID/var/lib/apt
zfs create rpool/ROOT/ubuntu_$UUID/var/lib/dpkg
zfs create rpool/ROOT/ubuntu_$UUID/var/lib/NetworkManager
zfs create rpool/ROOT/ubuntu_$UUID/var/log
zfs create rpool/ROOT/ubuntu_$UUID/var/mail
zfs create rpool/ROOT/ubuntu_$UUID/var/snap
zfs create rpool/ROOT/ubuntu_$UUID/var/spool
```

```
zfs create rpool/ROOT/ubuntu_$UUID/var/www

zfs create -o canmount=off -o mountpoint=/ \
    rpool/USERDATA
zfs create -o com.ubuntu.zsys:bootfs-datasets=rpool/ROOT/ubuntu_$UUID \
    -o canmount=on -o mountpoint=/root \
    rpool/USERDATA/root_$UUID
```

Si on veut un ensemble de données séparé pour /tmp :

```
zfs create -o com.ubuntu.zsys:bootfs=no \
    rpool/ROOT/ubuntu_$UUID/tmp
chmod 1777 /mnt/tmp
```

L'objectif principal de cette disposition de jeu de données est de séparer le système d'exploitation des données utilisateur. Cela permet de restaurer le système de fichiers racine sans restaurer les données utilisateur.

Si on ne fait rien de plus, /tmp sera stocké dans le système de fichiers racine. on peut également créer un ensemble de données distinct pour /tmp, comme indiqué ci-dessus. Cela empêche les données /tmp des instantanés du système de fichiers racine. Il vous permet également de définir un quota sur rpool/tmp, si vous souhaiter limiter l'espace maximum utilisé. Sinon, on peut utiliser un tmpfs (système de fichiers RAM) plus tard.

Facultatif : Ignorer les requêtes synchrones :

Les cartes SD sont relativement lentes. Si vous souhaiter augmenter les performances (en particulier lors de l'installation de packages) au prix d'une certaine sécurité, on peut désactiver le vidage des requêtes synchrones (par exemple, fsync(), O_[D]SYNC) :

Choisir l'une des options suivantes :

- Pour le système de fichiers racine, mais pas les données utilisateur :

```
zfs set sync=disabled rpool/ROOT
```

- Pour tout:

```
zfs set sync=disabled rpool
```

ZFS est transactionnel, il sera donc toujours cohérent en cas de plantage. Cependant, il faut laisser la synchronisation à sa valeur par défaut si ce système doit garantir la persistance (par exemple, s'il s'agit d'une base de données ou d'un serveur NFS).

Copier le système dans les systèmes de fichiers ZFS :

```

```

```
(cd /; tar -cf - --one-file-system --warning=no-file-ignored .) | \
    pv -p -bs $(du -sxm --apparent-size / | cut -f1)m | \
    (cd /mnt ; tar -x)
```

Étape 4 : Configuration du système

Configurer le nom d'hôte :

Remplacer HOSTNAME par le nom d'hôte souhaité :

```
echo HOSTNAME > /mnt/etc/hostname
vi /mnt/etc/hosts
```

- Ajouter une ligne : `127.0.1.1 HOSTNAME`
- ou si le système a un vrai nom dans DNS : `127.0.1.1 HOSTNAME FQDN`

Arrêter zed :

```
systemctl stop zed
```

Chroot dans le système de fichier

Lier les systèmes de fichiers virtuels de l'environnement en cours d'exécution au nouvel environnement ZFS et se chrooter dedans :

```
mount --rbind /boot/firmware /mnt/boot/firmware
mount --rbind /dev  /mnt/dev
mount --rbind /proc /mnt/proc
mount --rbind /run  /mnt/run
mount --rbind /sys  /mnt/sys
chroot /mnt /usr/bin/env DISK=$DISK UUID=$UUID bash --login
```

Configurer un environnement système de base :

```
apt update
```

Même si on préfére une langue système autre que l'anglais, S'assurer toujours que `en_US.UTF-8` est disponible :

```
dpkg-reconfigure locales
dpkg-reconfigure tzdata
```

Pour les installations LUKS uniquement, configurer /etc/crypttab :

```
# cryptsetup est déjà installé, mais cela le marque comme manuellement
# installé afin qu'il ne soit pas automatiquement supprimé.
apt install --yes cryptsetup

echo luks1 UUID=$(blkid -s UUID -o value ${DISK}-part4) none \
    luks,discard,initramfs > /etc/crypttab
```

L'utilisation d'initramfs est une solution de contournement car cryptsetup ne prend pas en charge ZFS.

Facultatif : monter un tmpfs sur /tmp

Si on a choisi de créer un ensemble de données /tmp ci-dessus, ignorer cette étape, car ce sont des choix mutuellement exclusifs. Sinon, on peut mettre /tmp sur un tmpfs (système de fichiers RAM) en activant l'unité tmp.mount.

```
cp /usr/share/systemd/tmp.mount /etc/systemd/system/
systemctl enable tmp.mount
```

Patcher une dépendance pour le chiffrement natif ZFS ou LUKS :

```
curl
https://launchpadlibrarian.net/478315221/2150-fix-systemd-dependency-loops.p
atch | \
    sed "s|/etc|/lib|;s|\.in$||" | (cd / ; sudo patch -p1)
```

Ce correctif provient du bogue #1875577 L'échange crypté ne se chargera pas le 20.04 avec la racine zfs.

Correction de l'ordre de montage du système de fichiers :

Il faut activer zfs-mount-generator. Cela rend systemd conscient des points de montage séparés, ce qui est important pour des choses comme /var/log et /var/tmp. À son tour, rsyslog.service dépend de var-log.mount via local-fs.target et les services utilisant la fonction PrivateTmp de systemd utilisent automatiquement After=var-tmp.mount.

```
mkdir /etc/zfs/zfs-list.cache
touch /etc/zfs/zfs-list.cache/rpool
ln -s /usr/lib/zfs-linux/zed.d/history_event-zfs-list-cacher.sh
/etc/zfs/zed.d
zed -F &
```

Forcer une mise à jour du cache :

```
zfs set canmount=noauto rpool/ROOT/ubuntu_$UUID
```

Vérifier que zed a mis à jour le cache en vous assurant qu'il n'est pas vide, ce qui prendra quelques secondes :

```
cat /etc/zfs/zfs-list.cache/rpool
```

Arrêter zed :

```
fg
Press Ctrl-C.
```

Corriger les chemins pour éliminer /mnt :

```
sed -Ei "s|/mnt/?|/|" /etc/zfs/zfs-list.cache/*
```

Supprimer l'ancien système de fichiers de /etc/fstab :

```
vi /etc/fstab
# Supprimer l'ancienne ligne du système de fichiers racine :
# LABEL=writable / ext4 ...
```

Configurer la ligne de commande du noyau :

```
cp /boot/firmware/cmdline.txt /boot/firmware/cmdline.txt.bak
sed -i "s|root=LABEL=writable
rootfstype=ext4|root=ZFS=rpool/ROOT/ubuntu_$UUID|" \
    /boot/firmware/cmdline.txt
sed -i "s| fixrtc||" /boot/firmware/cmdline.txt
sed -i "s|$| init_on_alloc=0|" /boot/firmware/cmdline.txt
```

Le script **fixrtc** n'est pas compatible avec ZFS et entraînera un blocage du démarrage pendant 180 secondes.

L'init*on*alloc=0 est destiné à traiter les régressions de performances.

Facultatif (mais fortement recommandé) : Faciliter le démarrage du débogage :

```
sed -i "s|$| nosplash|" /boot/firmware/cmdline.txt
```

Redémarrer :

```
exit
reboot
```

Attendre que le système nouvellement installé démarre normalement. Se connecter en tant qu'ubuntu et devenir root avec `sudo -i`.

Étape 5 : premier démarrage

Supprimer la partition ext4 et développer la partition ZFS :

```
sfdisk /dev/mmcblk0 --delete 3
echo ", +" | sfdisk --no-reread -N 2 /dev/mmcblk0
```

cela n'étend pas automatiquement le pool. Cela se produira au redémarrage.

Créer un compte utilisateur :

Remplacer le nom d'utilisateur par le nom d'utilisateur souhaité :

```
UUID=$(dd if=/dev/urandom bs=1 count=100 2>/dev/null |
    tr -dc 'a-z0-9' | cut -c-6)
ROOT_DS=$(zfs list -o name | awk '/ROOT\/ubuntu_/{print $1;exit}')
zfs create -o com.ubuntu.zsys:bootfs-datasets=$ROOT_DS \
    -o canmount=on -o mountpoint=/home/username \
    rpool/USERDATA/username_$UUID
adduser username

cp -a /etc/skel/. /home/username
chown -R username:username /home/username
usermod -a -G adm,cdrom,dip,lxd,plugdev,sudo username
```

Redémarrer :

```
reboot
```

Attendre que le système démarre normalement. Se connecter avec son nom d'utilisateur et devenir root avec `sudo -i`.

Développer le pool ZFS :

Vérifier que le pool a été étendu :

```
zfs liste rpool
```

S'il ne s'est pas développé automatiquement, essayer de le développer manuellement :

```
zpool online -e rpool mmcblk0p2
```

Supprimer l'utilisateur ubuntu :

```
deluser --remove-home ubuntu
```

Étape 6 : Installation complète du logiciel

Facultatif : Supprimer cloud-init :

```
vi /etc/netplan/01-netcfg.yaml

 network:
   version: 2
   ethernets:
     eth0:
       dhcp4: true

rm /etc/netplan/50-cloud-init.yaml
apt purge --autoremove ^cloud-init
```

Facultatif : supprimer les autres packages de stockage :

```
apt purge --autoremove bcache-tools btrfs-progs cloud-guest-utils lvm2 \
    mdadm multipath-tools open-iscsi overlayroot xfsprogs
```

Mettre à niveau le système minimal :

```
apt dist-upgrade --yes
```

Facultatif : installer un environnement GUI complet :

```

```

```
apt install --yes ubuntu-desktop
```

Si on installe un environnement GUI complet, il est préférable de supprimer cloud-init comme indiqué ci-dessus, pour gérer le réseau avec NetworkManager :

```
rm /etc/netplan/*.yaml
vi /etc/netplan/01-network-manager-all.yaml

network:
version: 2
renderer: NetworkManager
```

Facultatif (mais recommandé) : désactiver la compression des journaux :

Comme `/var/log` est déjà compressé par ZFS, la compression de logrotate va brûler les E/S du processeur et du disque pour (dans la plupart des cas) très peu de gain. De plus, si on crée des instantanés de `/var/log`, la compression de **logrotate** gaspillera en fait de l'espace, car les données non compressées vivront dans l'instantané. on peut éditer les fichiers dans `/etc/logrotate.d` à la main pour commenter la compression, ou utiliser cette boucle (copier-coller fortement recommandé) :

```
for file in /etc/logrotate.d/* ; do
    if grep -Eq "(^|[^#y])compress" "$file" ; then
        sed -i -r "s/(^|[^#y])(compress)/\1#\2/" "$file"
    fi
done
```

Redémarrer :

```
reboot
```

Étape 7 : Nettoyage final

Attendre que le système démarre normalement. Se connecter en utilisant le compte créé. S'assurer que le système (y compris le réseau) fonctionne normalement.

Facultatif : pour les installations LUKS uniquement, sauvegarder l'en-tête LUKS :

```
sudo cryptsetup luksHeaderBackup /dev/disk/by-id/scsi-SATA_disk1-part4 \
    --header-backup-file luks1-header.dat
```

Stocker cette sauvegarde dans un endroit sûr. Il est protégé par votre mot de passe LUKS, mais on peut utiliser un cryptage supplémentaire.