Installation FreeBSD sur un root ZFS

Table of Contents

Cet article présente l'installation de FreeBSD sur un root ZFS.

ZFS sur FreeBSD est un système de fichiers rapide, sécurisé et flexible. L'une des limitations actuelles lors de l'installation de FreeBSD 9 est qu'on ne peut pas installer le système d'exploitation sur un pool racine ZFS à partir du programme d'installation par défaut.

Cet article explique la méthode la plus simple pour installer une image de clé USB FreeBSD sur une clé USB et éffectuer une installation à partir de la clé. En utilisant cette méthode, on peut faire une installation complète de FreeBSD sur une racine ZFS en environ 2 minutes. La vitesse est intéressante, mais le véritable avantage est qu'on peut faire une installation, jouer avec l'image

pendant un moment et réinstaller sans se soucier de perdre beaucoup de temps à réinstaller.

Créer une image de clé USB

Il faut quelques étapes pour obtenir une image FreeBSD sur une clé USB et enfin mettre le script dessus, mais la tâche n'est pas difficile. Le côté positif est qu'une fois qu'on a créé la clé, on peut l'utiliser autant de fois qu'on le souhaite et enregistrer les modifications apportées à la clé lorsqu'on modifie les configurations. C'est un avantage qu'une installation sur CD ou DVD n'offre pas facilement.

Étape 1 : Télécharger l'image FreeBSD et ajouter l'image sur la clé USB

- Méthode 1: image memstick

FreeBSD est proposé en plusieurs version dont une version memstick, mais cela nécessite quelques adaptations pour le Raspberry4:

Télécharger la dernière version de l'image memstick `curl -4JOL https://download.freebsd.org/ftp/releases/amd64/amd64/ISO-IMAGES/12.0/FreeBSD-12.0-RELEASE-amd64-memstick.img`.

Graver l'image avec l'une des méthodes décrites ici.

Télécharger les fichiers suivants sur https://github.com/raspberrypi/firmware/tree/master/bootet et les remplacer dans la partition MSDOS de la clé USB:

1. fixup4.dat
2. start4.elf
3. bcm2711-rpi-4-b.dtb

Télécharger `u-boot.bin` à partir de l'emplacement suivant et le remplacer dans la clé USB: https://sourceforge.net/projects/rpi4uboot202010-fbsdonly-klaus/files/u-boot.bin/download.

Des étapes supplémentaires sont requises pour le PI4 8 Go:
1-Récupérer une toolchain de cross compile aarch64-unknown-linux-gnu (par exemple https://releases.linaro.org/components/toolchain/binaries/4.9-2017.01/aarch64-linux-gnu/gcc-linaro-4.9.4-2017.01-x86_64_aarch64-linux-gnu.tar.xz)
2-Vérifier les paramètres de configuration du noyau (par exemple, activer USB Attached SCSI dans le noyau si un périphérique de stockage USB3 est utilisé CONFIG_USB_UAS pour activer la prise en charge de SCSI connecté par USB3.)
3-Construire le noyau `$ARCH=arm64 CROSS_COMPILE=aarch64-unknown-linux-gnu- make bcm2711_defconfig`
4-Installer l'arborescence des périphériques et les superpositions en exécutant make dtb `$ARCH=arm64 CROSS_COMPILE=aarch64-unknown-linux-gnu- make dtb`
5-copier le fichier bcm2711-rpi-4-b.dtb dans /boot `cp -v arch/arm64/boot/dts/broadcom/bcm2711-rpi-4-b.dtb /mnt/xxxx/boot/`

Éjecter la clé USB et l'insérer dans le Pi (s'assurer qu'il n'y a pas de carte MicroSD dans le Pi avant de brancher), le Pi devrait démarrer sur la clé USB

- Méthode 2: image ISO RPI

FreeBSD est proposé en plusieurs version dont une version memstick, malheureusement les nouvelles versions de FreeBSD ne prennent plus en charge le démarrage USB, il accroche juste. Cependant memstick est l'image officielle d'installation de FreeBSD, elle comporte seule l'ensemble des fichiers nécessaires.

Pour récupérer ces fichiers d'installation il faut télécharger l'image memstick et extraire le dossier `/usr/freebsd-dist/`, on peut également télécharger les fichiers depuis le dépôt officiel:

```
mkdir -p freebsd-dist && cd freebsd-dist
curl -O
https://download.freebsd.org/ftp/releases/arm64/13.0-RELEASE/kernel
.txz
curl -O
https://download.freebsd.org/ftp/releases/arm64/13.0-RELEASE/base.t
xz
curl -O
https://download.freebsd.org/ftp/releases/arm64/13.0-RELEASE/MANIFE
ST
```

Télécharger la dernière version de l'instantané RPI3 (aucune version RPI4 n'est encore disponible). https://download.freebsd.org/ftp/snapshots/arm64/aarch64/ISO-IMAGES/13.0/FreeBSD-13.0-CURRENT-arm64-aarch64-RPI3-20201105-ef87bd449eb.img.xz, mais le release 20201112 fonctionne aussi.

Cette image est compressée, il faut donc commencer par la décompresser :

```
unxz FreeBSD-12.0-RELEASE-arm64-aarch64-RPI3.img.xz
```

Et on obtient le fichier FreeBSD .img. C'est lui que l'on doit transférer sur la carte SD.

Il est bien sûr nécessaire d'avoir une machine qui comporte un lecteur de carte SD interne ou externe.

Graver l'image sur la carte SD avec l'une des méthodes décrites ici.

L'image ISO utilise la méthode de boot UEFI, on ne peut donc utiliser que cette méthode lors de la configuration de la partition de boot.

Étape 2 : Démarrer la clé USB

Lorsque l'image a été écrite sur la clé USB, on peut maintenant démarrer le Raspberry Pi.

Brancher la clé USB et allumer le Raspberry Pi. On doit voir un écran multicolore (qui indique que le bootloader intégré au CPU lit les données de la partition SD/USB) puis le logo Raspberry Pi noir et blanc une fois le firmware UEFI prêt.

A ce stade, on peut appuyer sur Echap pour entrer dans la configuration du firmware, F1 pour lancer le Shell UEFI, ou, à condition d'avoir également un bootloader UEFI sur la carte SD ou sur une clé USB dans efi/boot/bootaa64.efi, on peut laisser le système UEFI l'exécuter (ce qui sera la valeur par défaut si aucune action n'est entreprise).

Étape 3 : Préparer l'environnement d'installation

memstick est un image read only, pour pouvoir téléchager des fichiers et les enregistrer de manière persistante sur la clé:
* remonter le système de fichiers racine sur la clé pour lire et écrire: `mount -rw /`
* activer la mise en réseau : `dhclient igb0`

Si le répertoire `/usr/freebsd-dist` n'existe pas (dans le cas d'une installation depuis ISO RPI) télécharger le fichiers d'installation récupérés à l'étape précédente:

```
cd ~/freebsd-dist
scp kernel.txz root@192.168.1.99:/usr/freebsd-dist/
scp base.txz root@192.168.1.99:/usr/freebsd-dist/
scp MANIFEST root@192.168.1.99:/usr/freebsd-dist/
```

Si on dispose du réseau, on peut télécharger directement les fichiers depuis le dépôt officiel:

```
mkdir -p /usr/freebsd-dist && cd /usr/freebsd-dist
fetch
https://download.freebsd.org/ftp/releases/arm64/13.0-RELEASE/kernel
.txz
fetch
https://download.freebsd.org/ftp/releases/arm64/13.0-RELEASE/base.t
xz
fetch
https://download.freebsd.org/ftp/releases/arm64/13.0-RELEASE/MANIFE
ST
```

Installation FreeBSD sur une racine ZFS

Etape 1: supprimer toutes les anciennes partitions sur le lecteur de destination

Pour refaire un partitionnement sur une carte déjà utilisée:

```
umount zroot || true
umount /mnt || true
zpool destroy zroot || true
gpart delete -i 2 mmcsd0 || true
gpart delete -i 1 mmcsd0 || true
```

```
gpart destroy -F mmcsd0 || true
```

Etape 2: Créer une partition de boot zfs (512k) et une partition racine de 220 Go

```
gpart create -s gpt mmcsd0
```

- Pour un boot Legacy

```
gpart add -a 4k -s 512k -t freebsd-boot mmcsd0
gpart bootcode -b /boot/pmbr -p /boot/gptzfsboot -i 1 mmcsd0
```

- Pour un boot UEFI

u-boot, doit charger `/boot/loader.efi` (copié sur la partition EFI/EXFAT) et ce fichier chargera le noyau, lira loader.conf et définira les variables appropriées nécessaires au démarrage).

On peut renommer loader.efi (sur la partition EFI) en `efi\boot\bootaa64.efi` et il devrait être sélectionné automatiquement par **u-boot**.

Le micrologiciel UEFI s'exécute à la mise sous tension et recherche un chargeur de système d'exploitation dans la partition système EFI. Le chemin vers le chargeur peut être défini par une variable d'environnement EFI. S'il n'est pas défini, la valeur par défaut spécifique à l'architecture est utilisée:
- amd64 /EFI/BOOT/BOOTX64.EFI
- arm /EFI/BOOT/BOOTARM.EFI
- arm64 /EFI/BOOT/BOOTAA64.EFI

```
gpart add -a 4k -s 512K -t efi mmcsd0
# pour les versions antérieures à FreeBSD 12.x
gpart bootcode -p /boot/boot1.efifat -i 1 mmcsd0
# pour les versions FreeBSD 12.x et supérieures, il faut utiliser une
partition FAT32
newfs_msdos -F 32 -c 1 /dev/mmcsd0p1
mount -t msdosfs -o longnames /dev/mmcsd0p1 /mnt
mkdir -p /mnt/EFI/BOOT
cp /boot/loader.efi /mnt/EFI/BOOT/BOOTAA64.efi
```

- Créer les partitions swap et zfs

Ajouter des partitions à utiliser par FreeBSD. Une partition freebsd-swap et une partition freebsd-zfs.

- **-a <nombre>**: contrôle l'alignement.
- **-s <taille>**: définit la taille de la partition, si elle n'est pas définie, elle utilisera par défaut l'espace restant sur le disque.
- **-l <nom>**: définit l'étiquette de la partition

```
gpart add -a 1m -s 4G -t freebsd-swap -l swap0 mmcsd0
gpart add -a 1m -t freebsd-zfs -l disk0 mmcsd0
```

Lors de la création des partitions une message signalera qu'elles ne sont pas allignéees, lorsque la taille du secteur de la partion n'est pas alignée, la vitesse de lecture peut être altérée.

Il est possible d'attribuer explicitement la taille du secteur lorsque les périphériques sont ajoutés pour la première fois à un pool (généralement au moment de la création du pool ou lors de l'ajout d'un vdev au pool). La valeur de ashift est en fait une valeur de décalage de bit allant de 9 à 16, la valeur par défaut 0 signifiant que ZFS doit détecter automatiquement la taille du secteur, la valeur de **ashift** pour 512 octets est 9 (2^9 = 512) tandis que la valeur de ashift pour 4 096 octets est 12 (212 = 4 096).

Pour déterminer la valeur de décalage, utiliser la commande `diskinfo -v mmcsd0 | grep stripesize`. Par exemple, pour un stripsize de 4096 le besoin d'un décalage est de 12 (2^12=4096). La valeur de décalage par défaut est 9 (2^9=512).

Pour forcer le pool à utiliser 4 096 secteurs d'octets au moment de la création du pool il faut utiliser gnop[1)]:

```
gnop create -S 4096 /dev/gpt/disk0`</WRAP>
```

Etape 3: Créer le zpool

```
mount -t tmpfs tmpfs /mnt
zpool create -f -o altroot=/mnt -o cachefile=/var/tmp/zpool.cache zroot
/dev/gpt/disk0.nop
zpool export zroot
gnop destroy /dev/gpt/disk0.nop
```

```
zpool import -o altroot=/mnt -o cachefile=/var/tmp/zpool.cache zroot
```

Etape 4: Définir la propriété bootfs et définir les options

```
zpool set bootfs=zroot zroot
zpool set listsnapshots=on zroot
zfs set logbias=throughput zroot
zfs set compression=lz4 zroot
zfs set atime=off zroot
zfs set copies=2 zroot
```

Etape 5: Ajouter un espace d'échange et appliquer des options

```
zfs create -V 2G zroot/swap
zfs set org.freebsd:swap=on zroot/swap
zfs set copies=1 zroot/swap
```

Etape 6: Créer un lien symbolique vers /home et corriger certaines autorisations

```
cd /mnt/zroot ; ln -s usr/home home
```

Etape 7: Configuration de la mise en cache en écriture

ZFS, comme la plupart des autres systèmes de fichiers, essaie de conserver un tampon d'opérations d'écriture en mémoire, puis de l'écrire sur les disques au lieu de l'écrire directement sur les disques. C'est ce que l'on appelle l' écriture asynchrone et elle donne des gains de performances pour les applications tolérantes aux pannes ou où la perte de données ne fait pas beaucoup de dégâts. Le système d'exploitation stocke simplement les données en mémoire et indique à l'application, qui a demandé l'écriture, que l'écriture est terminée. Il s'agit du comportement par défaut de nombreux systèmes d'exploitation, même lors de l'exécution de ZFS.

Cependant, le fait demeure qu'en cas de panne du système ou de panne de courant, toutes les écritures tamponnées dans la mémoire principale sont perdues. Ainsi, les applications qui souhaitent une cohérence par rapport aux performances peuvent ouvrir des fichiers en mode synchrone , puis les données ne sont considérées comme écrites qu'une fois qu'elles sont réellement sur le disque. La plupart des bases de données et des applications comme NFS reposent en permanence sur des écritures synchrones.

On pet définir l'indicateur: `sync = always` pour faire des écritures synchrones le comportement par défaut pour tout ensemble de données donné. `$zfs set sync=always mypool/dataset1`

Pour mettre en cache les données pour des écritures plus longues on peut utiliser

`vfs.zfs.txg.timeout`[2] pour retarder la validation des écritures, par exemple `vfs.zfs.txg.timeout="60"` permet de valider les écritures asynchrones après 1 minute si la limite d'écriture n'a pas été atteinte.

```
sysctl vfs.zfs.min_auto_ashift=12
sysctl vfs.zfs.trim.txg_batch=128
sysctl vfs.zfs.txg.timeout=60
sysctl vfs.zfs.vdev.def_queue_depth=128
sysctl vfs.zfs.vdev.write_gap_limit=0
```

Bien sûr, on peut souhaiter privilégier la sécurité, il faut alors réduire la valeur par défaut de 30s à 5s par exemple: `vfs.zfs.txg.timeout="5"`.

Etape 8: Installer FreeBSD .

```
cd /usr/freebsd-dist
cat base.txz | tar --unlink -xpJf - -C /mnt/zroot/
cat kernel.txz | tar --unlink -xpJf - -C /mnt/zroot/
cat ports.txz | tar --unlink -xpJf - -C /mnt/zroot/
cat src.txz | tar --unlink -xpJf - -C /mnt/zroot/
```

Etape 9: Copier zpool.cache sur le disque d'installation

```
cp /var/tmp/zpool.cache /mnt/zroot/boot/zfs/zpool.cache
```

Etape 10: Configuration de boot/loader.conf

Le fichier loader.conf contient les informations d'amorçage du système. Grâce à lui, on peut spécifier le noyau à démarrer, les paramètres à lui transmettre, et les modules supplémentaires à charger ; ou plus généralement définir toutes les variables décrites dans loader.

- Configurer le montage racine de ZFS

Pour arm64, `zfs_load="YES"` n'ajoute pas opensolaris.ko en tant que kld dépendant, alors il faut ajouter explicitement `opensolaris_load="YES"` à loader.conf lors de l'installation du système sur un système de fichiers racine ZFS.

```
echo 'zfs_enable="YES"' >> /mnt/zroot/etc/rc.conf
echo 'zfs_load="YES"' >> /mnt/zroot/boot/loader.conf
```

```
echo 'opensolaris_load="YES"' >> /mnt/zroot/boot/loader.conf
echo 'vfs.root.mountfrom="zfs:zroot"' >> /mnt/zroot/boot/loader.conf
```

- Configurer la mise en cache en lecture

ARC (Adaptive Replacement Cach) est un algorithme moderne pour la mise en cache des données dans la DRAM. En d'autres termes, ARC n'est rien, mais il contient des données mises en cache telles que des données de système de fichiers et des métadonnées. ZFS essaiera d'utiliser autant de RAM en cache pour accélérer les opérations du serveur.

Si on envisage d'utiliser la déduplication et que la machine est sous-spécifiée, il faut définir `vfs.zfs.arc_max` sur une valeur raisonnable ou ZFS preandra autant de mémoire disponible que possible, ce qui peut créer des scénarios de manque de mémoire.

En supposant Ram=8 Go de mémoire les règlages suivant permettent d'optimiser la mise en cache:

```
echo 'vfs.zfs.arc_min="1024M"' >> /mnt/boot/loader.conf
echo 'vfs.zfs.arc_max="3584M"' >> /mnt/boot/loader.conf
```

- Désactiver la prélecture ZFS

La prélecture ZFS augmente la vitesse globale de ZFS, mais lorsque le vidage/écriture du disque se produit, le système est moins réactif (en raison des E/S disque extrêmes).

Les systèmes avec 4 Go de RAM ou plus ont la prélecture activée par défaut.

```
echo 'vfs.zfs.prefetch_disable="1"' >> /mnt/boot/loader.conf
```

- Augmenter le nombre de vnodes

Un vnode est la représentation interne d'un fichier ou d'un répertoire. Ainsi, l'augmentation du nombre de nœuds virtuels disponibles pour le système d'exploitation réduit les E/S de disque. Normalement, cela est géré par le système d'exploitation et n'a pas besoin d'être modifié. Dans certains cas où les E/S disque constituent un goulot d'étranglement et que le système manque de vnodes, ce paramètre devra être augmenté. La quantité de RAM inactive et libre devra être prise en compte. Le maximum par défaut est un peu plus de 200 000. Lorsque numvnodes atteint maxvnode, les performances diminuent considérablement, il faut donc jouer la sécurité en augmentant ce plafond.

```
echo 'kern.maxvnodes=250000' >> /mnt/boot/loader.conf
```

- Définir la limite d'écriture TXG sur un seuil inférieur.

Cela aide à “niveler” le débit. Une valeur de 256 Mo fonctionne bien pour les systèmes avec 4 Go de RAM, tandis que 1 Go fonctionne bien pour nous avec 8 Go sur disques disposant de 64 Mo de cache.

Dans la version 27 ou inférieure, ce paramètre s'appelle 'vfs.zfs.txg.write*limit*override'.

```
echo 'vfs.zfs.write_limit_override=1073741824' >> /mnt/boot/loader.conf
```

- Forcer l'utilisation des identifiants gpt au lieu des identifiants gptid ou des disques

```
echo 'kern.geom.label.disk_ident.enable="0"' >> /mnt/zroot/boot/loader.conf
echo 'kern.geom.label.gpt.enable="1"' >> /mnt/zroot/boot/loader.conf
echo 'kern.geom.label.gptid.enable="0"' >> /mnt/zroot/boot/loader.conf
```

- Activer la mise en réseau, pf et ssh et arrêter l'écoute de syslog.

```
echo 'hostname="FreeBSDzfs"' >> /mnt/zroot/etc/rc.conf
echo 'ifconfig_igb0="dhcp"' >> /mnt/zroot/etc/rc.conf
echo '#ifconfig_igb0="inet 192.168.0.150 netmask 255.255.255.0 ether
00:11:22:33:44:55"' >> /mnt/zroot/etc/rc.conf
echo '#defaultrouter="192.168.0.1"' >> /mnt/zroot/etc/rc.conf
echo '#pf_enable="YES"' >> /mnt/zroot/etc/rc.conf
echo '#pflog_enable="YES"' >> /mnt/zroot/etc/rc.conf
echo 'sshd_enable="YES"' >> /mnt/zroot/etc/rc.conf
echo 'syslogd_flags="-ss"' >> /mnt/zroot/etc/rc.conf
echo 'nameserver 1.1.1.1' >> /mnt/zroot/etc/resolv.conf
```

- drop des paquets envoyés aux ports fermés.

```
echo 'net.inet.icmp.drop_redirect=1' >> /mnt/zroot/etc/sysctl.conf
echo 'net.inet.sctp.blackhole=2' >> /mnt/zroot/etc/sysctl.conf
echo 'net.inet.tcp.blackhole=2' >> /mnt/zroot/etc/sysctl.conf
echo 'net.inet.tcp.drop_synfin=1' >> /mnt/zroot/etc/sysctl.conf
echo 'net.inet.tcp.path_mtu_discovery=0' >> /mnt/zroot/etc/sysctl.conf
echo 'net.inet.udp.blackhole=1' >> /mnt/zroot/etc/sysctl.conf
```

- Désactiver les connexions root distantes dans sshd

```
echo 'PermitRootLogin no' >> /mnt/zroot/etc/ssh/sshd_config
echo 'PermitEmptyPasswords no' >> /mnt/zroot/etc/ssh/sshd_config
```

- Désactiver sendmail dans etc/rc.conf

```
echo 'dumpdev="NO"' >> /mnt/zroot/etc/rc.conf
echo 'sendmail_enable="NONE"' >> /mnt/zroot/etc/rc.conf
```

Etape 11: Créer le /etc/fstab

Le fichier /etc/fstab doit être présent sinon freebsd ne démarrera pas correctement

```
touch /mnt/zroot/etc/fstab
```

Etape 11: Synchronisation... Installation terminée.

Eteindre, retirer la clé USB et redémarrer la machine.

```
sync
```

Ajouter ensuite un utilisateur privilégié au groupe 'wheel', pour pouvoir se connecter en ssh en tant que nouvel utilisateur et configurer la box.

Script d'installation racine ZFS pour FreeBSD 12.0

Chaque section du script est commentée, et peut être adaptée en fonction des besoins.

Le script zfs.sh ci-dessous reprend les étapes de l'installation:

1. supprimer toutes les partitions du lecteur. Ceci est nécessaire si on utilise le même script d'installation sur le même lecteur et qu'on ne fait que tester des installations. On veut nous assurer d'avoir une partition propre à chaque fois.
2. Ensuite, il crée une nouvelle table de partition et aligne le disque pour les secteurs 4K (4096 octets).
3. Le pool ZFS est créé et les options utiles sont définies:
    1. désactivation du bit de temps d'accès
    2. définition de la somme de contrôle sur fletcher4 et demande à ZFS de conserver deux (2) copies de chaque fichier (pour permettre à ZFS de réparer automatiquement les fichiers s'ils sont corrompus avec un impact minimal sur les performances).
4. Installation de FreeBSD à partir de l'image memstick
5. activation du client DHCP pour démarrer le réseau au démarrage, activation de ssh et désactivation des connexions root et de sendmail.

Une fois le script terminé, FreeBSD sera entièrement amorçable à partir d'un pool ZFS sur le disque dur cible.

```
#!/bin/sh
set -euf
```

```
#
# Calomel.org
#     https://calomel.org/zfs_freebsd_root_install.html
#     FreeBSD 13.0-RELEASE ZFS Root Install script
#     zfs.sh @ Version 0.26

# NOTE: mmcsd0 for SATA , nvd0 for PCIe M.2 NVMe

echo "# remove any old partitions on destination drive"
umount zroot || true
umount /mnt || true
zpool destroy zroot || true
gpart delete -i 2 mmcsd0 || true
gpart delete -i 1 mmcsd0 || true
gpart destroy -F mmcsd0 || true

echo ""
echo "# Create zfs boot (512k) and a 220 gig root partition"
gpart create -s gpt mmcsd0
gpart add -a 4k -s 512k -t freebsd-boot mmcsd0
gpart add -a 4k -s 220G -t freebsd-zfs -l disk0 mmcsd0
gpart bootcode -b /boot/pmbr -p /boot/gptzfsboot -i 1 mmcsd0

echo ""
 # Option 1: align to 4K, ashift=12
 echo "# Align the Disks for 4K (ashift=12) and create the pool"
 gnop create -S 4096 /dev/gpt/disk0

 # Option 2: align to 8k, ashift=13
#echo "# Align the Disks for 8K (ashift=13) and create the pool"
#gnop create -S 8192 /dev/gpt/disk0

zpool create -f -o altroot=/mnt -o cachefile=/var/tmp/zpool.cache zroot
/dev/gpt/disk0.nop
zpool export zroot
gnop destroy /dev/gpt/disk0.nop
zpool import -o altroot=/mnt -o cachefile=/var/tmp/zpool.cache zroot

echo ""
echo "# Set the bootfs property and set options"
zpool set bootfs=zroot zroot
zpool set listsnapshots=on zroot
zfs set logbias=throughput zroot
zfs set compression=lz4 zroot
zfs set atime=off zroot
zfs set copies=2 zroot

echo ""
echo "# Add swap space and apply options"
zfs create -V 2G zroot/swap
zfs set org.freebsd:swap=on zroot/swap
```

```
zfs set copies=1 zroot/swap

echo ""
echo "# Create a symlink to /home and fix some permissions"
cd /mnt/zroot ; ln -s usr/home home

echo ""
echo "# Set zfs to cache data for longer striped writes"
sysctl vfs.zfs.min_auto_ashift=12
sysctl vfs.zfs.trim.txg_batch=128
sysctl vfs.zfs.txg.timeout=900
sysctl vfs.zfs.vdev.def_queue_depth=128
sysctl vfs.zfs.vdev.write_gap_limit=0

echo ""
echo "# Install FreeBSD OS from *.txz memstick."
echo "# This will take a few minutes..."
cd /usr/freebsd-dist
export DESTDIR=/mnt/zroot

 # Option 1: install a 64bit os, no 32bit libs or ports or source
  for file in base.txz kernel.txz doc.txz;

 # Option 2: only install a 64bit os, no 32bit libs
 #for file in base.txz kernel.txz doc.txz ports.txz src.txz;

 # Option 3: full freebsd install
 #for file in base.txz lib32.txz kernel.txz doc.txz ports.txz src.txz;

do (cat $file | tar --unlink -xpJf - -C ${DESTDIR:-/}); done

echo ""
echo "# Copy zpool.cache to install disk."
cp /var/tmp/zpool.cache /mnt/zroot/boot/zfs/zpool.cache

echo ""
echo "# Setup ZFS root mount and boot"
echo 'zfs_enable="YES"' >> /mnt/zroot/etc/rc.conf
echo 'zfs_load="YES"' >> /mnt/zroot/boot/loader.conf
echo 'vfs.root.mountfrom="zfs:zroot"' >> /mnt/zroot/boot/loader.conf

echo ""
echo "# use gpt ids instead of gptids or disks idents"
echo 'kern.geom.label.disk_ident.enable="0"' >> /mnt/zroot/boot/loader.conf
echo 'kern.geom.label.gpt.enable="1"' >> /mnt/zroot/boot/loader.conf
echo 'kern.geom.label.gptid.enable="0"' >> /mnt/zroot/boot/loader.conf

echo ""
echo "# enable networking, pf and ssh and stop syslog from listening."
echo 'hostname="FreeBSDzfs"' >> /mnt/zroot/etc/rc.conf
echo 'ifconfig_igb0="dhcp"' >> /mnt/zroot/etc/rc.conf
```

```
echo '#ifconfig_igb0="inet 192.168.0.150 netmask 255.255.255.0 ether
00:11:22:33:44:55"' >> /mnt/zroot/etc/rc.conf
echo '#defaultrouter="192.168.0.1"' >> /mnt/zroot/etc/rc.conf
echo '#pf_enable="YES"' >> /mnt/zroot/etc/rc.conf
echo '#pflog_enable="YES"' >> /mnt/zroot/etc/rc.conf
echo 'sshd_enable="YES"' >> /mnt/zroot/etc/rc.conf
echo 'syslogd_flags="-ss"' >> /mnt/zroot/etc/rc.conf
echo 'nameserver 1.1.1.1' >> /mnt/zroot/etc/resolv.conf

echo ""
echo "# drop packets sent to closed ports."
echo 'net.inet.icmp.drop_redirect=1' >> /mnt/zroot/etc/sysctl.conf
echo 'net.inet.sctp.blackhole=2' >> /mnt/zroot/etc/sysctl.conf
echo 'net.inet.tcp.blackhole=2' >> /mnt/zroot/etc/sysctl.conf
echo 'net.inet.tcp.drop_synfin=1' >> /mnt/zroot/etc/sysctl.conf
echo 'net.inet.tcp.path_mtu_discovery=0' >> /mnt/zroot/etc/sysctl.conf
echo 'net.inet.udp.blackhole=1' >> /mnt/zroot/etc/sysctl.conf

echo ""
echo "# sshd, disable remote root logins."
echo 'PermitRootLogin no' >> /mnt/zroot/etc/ssh/sshd_config
echo 'PermitEmptyPasswords no' >> /mnt/zroot/etc/ssh/sshd_config

echo ""
echo "# /etc/rc.conf disable sendmail"
echo 'dumpdev="NO"' >> /mnt/zroot/etc/rc.conf
echo 'sendmail_enable="NONE"' >> /mnt/zroot/etc/rc.conf

echo ""
echo "# touch the /etc/fstab else freebsd will not boot properly"
touch /mnt/zroot/etc/fstab

sync
echo ""
echo "# Synchronisation... Installation terminée."
echo ""
echo "# Astuce : éteindre, retirer la clé USB et redémarrer la machine."
echo "# Ajouter ensuite un utilisateur privilégié au groupe 'wheel', pour"
echo "# pouvoir se connecter en ssh en tant que nouvel utilisateur et
configurer la box."
echo ""
sync

#### EOF ####
```

Étape 1 : Transférer le script zfs.sh

Copier le script dans le système de fichier en utilisant scp ou ssh, ou télécharger le script directement depuis calomel.org, puisqu'on est sur le réseau et qu'on a une adresse IP, pour cela utiliser fetch pour collecter le script,

```
fetch --no-verify-peer https://calomel.org/zfs.sh
```

A ce stade, on dispose d'une clé USB amorçable avec le script d'installation racine zfs.sh ZFS.

Lorsqu'on redémarre sur la clé, le système de fichiers revient en mode lecture seule.

Étape 2 : Rendre le script exécutable

Pour utiliser le script “zfs.sh”, il faut rendre ce fichier exécutable ou utiliser “sh zfs.sh” pour l'exécuter.

```
chmod 755 zfs.sh
```

C'est maintenant le moment pour regarder le script et apporter des modifications nécessaires (vi zfs.sh):
- La taille zroot par défaut du script est “220G” pour 220 gigaoctets qui s'adapte facilement sur un SSD de 240 gigaoctets. Modifier le script pour adapter le zroot plus grand. Utiliser l'étiquette « G » pour les gigaoctets et « T » pour les téraoctets (Le script générera une erreur si la partition est plus grande que le disque ne peut le prendre en charge)
- choisir l'option d'alignement du disque pour les secteurs 4K (4096 octets) ou 8K (8192 octets)
- choisir l'option d'installation de FreeBSD (Option 1=minimal, Option 2=réduit, Option 3=full)

Etape 3: Utilisation du script zfs.sh du memstick

Une fois qu'on a le script sur le memstick, l'installation de FreeBSD sur un disque dur ou SSD est incroyablement facile. Voici les étapes :

- Connecter à la fois la clé USB et un disque dur à la machine
- Démarrer sur la clé USB dans FreeBSD (image memstick)
- Choisir “Shell” dans le menu d'installation de démarrage
- Monter la clé USB en lecture/écriture : “mount -rw /”
- Exécuter le script : “./zfs.sh”

Le script s'exécutera, détruira toutes les partitions précédentes, créera la nouvelle partition ZFS et installera FreeBSD sur le disque dans un seul pool ZFS. Une fois l'installation terminée, on peut éteindre la machine, retirer la clé USB et démarrer le lecteur en s'assurant qu'il démarre correctement.

on peut utiliser n'importe quel type de disque comme un SSD, un disque dur en rotation ou un RAID de n'importe quelle taille. Cela n'a pas d'importance car le script utilisera simplement tout l'espace sur le lecteur pour ZFS.

Si on a l'erreur “Impossible de monter '/mnt/zroot' : échec de création du point de montage”, il faut s'assurer que le lecteur USB est monté en lecture/écriture à l'aide de la commande “mount -rw /”.

Pour aller plus loin

Comment patcher et mettre à jour manuellement FreeBSD

Pour mettre à jour manuellement FreeBSD, utiliser un simple alias appelé “upgrayedd”. Ajouter l'alias à /root/.profile et exécuter en tant qu'utilisateur root sur la ligne de commande. “upgrayedd” téléchargera et mettra à jour tous les packages, puis recherchera les correctifs système, téléchargera les correctifs système s'ils sont disponibles et installera les correctifs. La tâche finale consiste à supprimer tous les anciens packages dans /var/db/freebsd-update/files/ car ces anciens packages ne sont pas supprimés par l'outil pkg. Noter l'ajout de “env PAGER=cat” pour vous assurer que “freebsd-update fetch” ne s'interrompt pas à l'aide de la commande “more”.

```
alias upgrayedd='/usr/sbin/pkg upgrade && echo "" && /usr/sbin/pkg
autoremove && echo "" && /usr/sbin/pkg audit -F ; echo "" && env PAGER=cat
freebsd-update fetch install; /usr/sbin/pkg clean -ay; for i in
/var/db/freebsd-update/files/* ; do rm "$i"; done'
```

Ce qui suit est la sortie de l'alias “upgrayedd” lorsque le système local est entièrement corrigé et à jour.

```
root@FreeBSDzfs:  upgrayedd

Updating FreeBSD repository catalogue...
FreeBSD repository is up-to-date.
All repositories are up-to-date.
Checking for upgrades (1 candidates): 100%
Processing candidates (1 candidates): 100%
Checking integrity... done (0 conflicting)
Your packages are up to date.

Checking integrity... done (0 conflicting)
Nothing to do.
```

```
vulnxml file up-to-date
0 problem(s) in the installed packages found.

src component not installed, skipped
Looking up update.FreeBSD.org mirrors... 4 mirrors found.
Fetching metadata signature for 12.0-RELEASE from update4.freebsd.org...
done.
Fetching metadata index... done.
Fetching 1 metadata files... done.
Inspecting system... done.
Preparing to download files... done.

No updates needed to update system to 12.0-RELEASE.
No updates are available to install.
Run '/usr/sbin/freebsd-update fetch' first.
Nothing to do.
```

Comment ajouter un deuxième disque de démarrage miroir

Après avoir configuré la nouvelle installation à l'aide du script racine ZFS ci-dessus, on peut ajouter un autre disque de type et de capacité similaires pour être un lecteur de démarrage en miroir. Quand l'un des disques de démarrage meurt, on utilise toutes les données du lecteur de démarrage sur le deuxième lecteur et ce deuxième lecteur est capable de démarrer tout seul.

La première étape consiste à installer un autre disque sur la machine. Puisque on a installé le système d'exploitation sur le premier disque du bus SATA avec le script ci-dessus, ce lecteur sera connu sous le nom de “mmcsd0”. Utiliser `dmesg | grep ada` pour examiner tous les lecteurs reconnus. Lors de l'ajout d'un deuxième lecteur connecté au deuxième port SATA, ce nouveau lecteur sera “ada1”.

Les commandes suivantes créeront deux partitions sur le nouveau lecteur et attacheront le lecteur au zroot actuel. ZFS créera automatiquement un pool de miroirs utilisant les deux lecteurs et les données du lecteur de démarrage d'origine seront dupliquées sur le deuxième lecteur.

NE PAS redémarrer la machine tant que le processus n'est pas terminé. Vérifier la progression à l'aide de “zpool status”.

La dernière étape consiste à ajouter le code de démarrage au nouveau lecteur miroir afin que lorsque le premier lecteur meurt, le deuxième lecteur puisse démarrer la machine. on peut ajouter le code de démarrage pendant la duplication des deux disques ou attendre la fin. Lorsque le processus de est terminé, l'installation est terminée. À ce stade, on peut s'assurer que le deuxième lecteur démarre lorsque le premier lecteur est débranché.

Le script appelé zfs_mirror.sh utilise également les mêmes commandes pour ajouter la clé USB amorçable pour créer des disques de rechange.

```
root@FreeBSDzfs:~ # gpart create -s gpt ada1
root@FreeBSDzfs:~ # gpart add -a 4k -s 512k -t freebsd-boot ada1
root@FreeBSDzfs:~ # gpart add -a 4k -s 1T -t freebsd-zfs ada1
root@FreeBSDzfs:~ # zpool attach zroot mmcsd0p2 ada1p2
```

Si on veut démarrer à partir du pool 'zroot', il faudra peut-être mettre à jour

Ajout du code de démarrage sur le disque nouvellement connecté « ada1 ».

En supposant qu'on utilisier le partitionnement GPT et que 'ada1' soit votre nouveau disque de démarrage on peut utiliser la commande suivante :

```
gpart bootcode -b /boot/pmbr -p /boot/gptzfsboot -i 1 ada1

root@FreeBSDzfs:~ # gpart bootcode -b /boot/pmbr -p /boot/gptzfsboot -i 1
ada1
root@FreeBSDzfs:~ # zpool status
  pool: zroot
 state: ONLINE
  scan: scrub repaired 0 in 0h0m with 0 errors on Tue Dec 25 10:20:30 2020
config:

        NAME           STATE     READ WRITE CKSUM
        zroot          ONLINE       0     0     0
          mirror-0     ONLINE       0     0     0
            mmcsd0p2     ONLINE       0     0     0
            ada1p2     ONLINE       0     0     0

errors: No known data errors
```

Comment ajouter un disque de secours au pool de démarrage en miroir

Une fois qu'on a un pool de démarrage ZFS en miroir, comme expliqué ci-dessus, on peut décider d'ajouter un disque de secours pour une assurance supplémentaire contre les pannes de disque. Si l'un des disques en miroir meurt, le disque de secours sera immédiatement ajouté au pool en miroir et une copie des données sera répliquée sur le disque de secours.

Tout d'abord, il faut s'assurer que le pool miroir est sain. On ajoute le troisième disque physique dans le châssis qui est ada2. ada2 a la même taille que les deux autres disques en miroir. Créer exactement le même schéma de partition qu'on a utilisé pour créer les lecteurs de démarrage en miroir et ajouter le code de démarrage au lecteur de secours. Enfin, ajouter ada2 au pool en miroir en tant que spare. Le dernier `zpool status` vérifie que ada2 est maintenant une réserve pour le pool zroot. on peut ajouter autant de spare qu'on le souhaite.

```
root@FreeBSDzfs:~ # zpool status
  pool: zroot
 state: ONLINE
  scan: scrub repaired 0 in 0h0m with 0 errors on Tue Dec 25 10:20:30 2020
config:

        NAME           STATE     READ WRITE CKSUM
        zroot          ONLINE       0     0     0
          mirror-0     ONLINE       0     0     0
            mmcsd0p2     ONLINE       0     0     0
            ada1p2     ONLINE       0     0     0

errors: No known data errors

root@FreeBSDzfs:~ # gpart destroy -F ada2
root@FreeBSDzfs:~ # gpart create -s gpt ada2
root@FreeBSDzfs:~ # gpart add -a 4k -s 512k -t freebsd-boot ada2
root@FreeBSDzfs:~ # gpart add -a 4k -s 1T -t freebsd-zfs ada2
root@FreeBSDzfs:~ # gpart bootcode -b /boot/pmbr -p /boot/gptzfsboot -i 1
ada2
root@FreeBSDzfs:~ # zpool add zroot spare ada2
root@FreeBSDzfs:~ # zpool status
  pool: zroot
 state: ONLINE
  scan: scrub repaired 0 in 0h0m with 0 errors on Tue Dec 25 10:20:30 2020
config:

        NAME           STATE     READ WRITE CKSUM
        zroot          ONLINE       0     0     0
          mirror-0     ONLINE       0     0     0
            mmcsd0p2     ONLINE       0     0     0
            ada1p2     ONLINE       0     0     0
          spares
            ada2       AVAIL

errors: No known data errors
```

Comment vérifier que ZFS est aligné 4k sur les disques Advanced Format

Advanced Format est un terme désignant tout format de secteur SSD ou disque dur moderne stockant des données sur un disque dépassant 512 à 520 octets par secteur, comme les secteurs de 4096 octets (4 Ko). Les secteurs 4k plus grands utilisent la surface de stockage plus efficacement pour les fichiers volumineux mais moins efficacement pour les fichiers plus petits, et permettent l'intégration d'algorithmes de correction d'erreurs plus puissants pour maintenir l'intégrité des données dans des zones de stockage plus élevées ités.

Pour que le lecteur soit correctement aligné, il faut vérifier à la fois que la partition est alignée sur 4k sur le disque et que ZFS est configuré pour écrire sur des secteurs de 4 096 octets.

Tout d'abord, il faut s'assurer que gpart a utilisé la directive “-a 4k” pour aligner la partition en 4k

comme dans le script d'installation racine ZFS ci-dessus. Utiliser “diskinfo” pour interroger la partition de disque et rechercher la valeur “stripeoffset”. Prendre ensuite le module du “stripeoffset” contre 4096 octets et, si le résultat est zéro(0), la partition est alignée 4k sur le disque. Voici un exemple de lecteur après avoir utilisé le script d'installation racine ZFS :

```
[root@zfsFBSD10 ~]# diskinfo -v mmcsd0p2
mmcsd0p2
        512             # sectorsize
        118111600640    # mediasize in bytes (110G)
        230686720       # mediasize in sectors
        0               # stripesize
---->   544768          # stripeoffset
        228855          # Cylinders according to firmware.
        16              # Heads according to firmware.
        63              # Sectors according to firmware.
        140879140000971400F7    # Disk ident.

[root@zfsFBSD10 ~]# echo 544768 % 4096 | bc
0

 (si le résultat est zéro (0) la partition est alignée sur 4k)
```

Deuxièmement, vérifier que ZFS écrit la plus grande taille de secteur de 4k. En utilisant l'outil “zdb”, on examine le pool zroot ZFS et trouve la valeur pour `ashift`. Il faut s'assurer que ashift est égal à douze (12), ce qui signifie 4 096 secteurs d'octets et NON à neuf (9), ce qui ne représente que 512 secteurs d'octets. La valeur de décalage pour 512 octets est 9 (2^9 = 512) tandis que la valeur de décalage pour 4 096 octets est de 12 (2^12 = 4 096).

ashift ne signifie PAS alignement, mais spécifie uniquement que ZFS est configuré pour utiliser des secteurs de 4 096 octets.

Ce qui suit est le résultat du même lecteur installé avec le script racine ZFS et on peut voir la valeur “ashift: 12”.

```
[root@zfsFBSD10 ~]# zdb
zroot:
    version: 5000
    name: 'zroot'
    state: 0
    txg: 11607
    pool_guid: 10810981494469860258
    hostid: 1232771827
    hostname: 'calomel'
    vdev_children: 1
    vdev_tree:
        type: 'root'
```

```
        id: 0
        guid: 10810981494469860258
        children[0]:
            type: 'disk'
            id: 0
            guid: 2636318580918461148
            path: '/dev/mmcsd0p2'
            phys_path: '/dev/mmcsd0p2'
            whole_disk: 1
            metaslab_array: 33
            metaslab_shift: 30
---->       ashift: 12
            asize: 118106882048
            is_log: 0
            DTL: 52
            create_txg: 4
    features_for_read:
```

L'installation de root avec compression améliore-t-elle les performances ?

Oui. La compression réduit la quantité de blocs écrits sur le disque dur, augmentant ainsi la densité aérienne du disque. Cela signifie qu'on peut lire et écrire des données sur le disque plus rapidement qu'avec des données brutes. Certains peuvent dire que la compression ralentira l'accès aux E/S. Ce n'est pas vrai car les processeurs d'aujourd'hui perdent la plupart de leur temps à attendre que le disque dur fasse son travail.

De plus, lors de l'installation de FreeBSD en utilisant lz4 sur un disque de démarrage SSD, on constate une économie de compression de 2,22x. Cela réduit la quantité de données écrites sur le SSD et, selon certains, augmente la durée de vie du disque SSD.

```
[root@zfsFBSD10 ~]# zfs get all zroot | grep compress
zroot  compressratio         2.22x                  -
zroot  compression           lz4                    local
zroot  refcompressratio      2.22x                  -
```

Comment créer une clé USB basée sur ZFS pour les données générales ?

Une clé USB ou une clé USB est un excellent moyen de déplacer des données dans le bureau ou pour des sauvegardes.

Les commandes suivantes utilisent gpart pour créer la partition alignée 4K et zpool pour créer un pool ZFS appelé “sauvegarde”. Lorsque la clé USB est montée, elle apparaîtra comme “/backup” . Le prochain ensemble de commandes zfs et zpool définira nos options de volume ZFS préférées sur la clé USB, y compris la compression et la conservation d'au moins deux (2) copies de chaque fichier pour la cohérence des données. Vous ne pouver jamais être trop prudent avec vos données.

Créer un pool ZFS sur une clé USB (4k aligné)

```
gpart destroy -F da0
gnop create -S 4096 da0
zpool create -f backup /dev/da0.nop
zpool export backup
gnop destroy /dev/da0.nop
zpool import backup
```

Options de volume ZFS

```
zfs set atime=off backup
zfs set compression=lz4 backup
zfs set copies=2 backup
zfs set logbias=throughput backup
zpool set listsnapshots=on backup
```

Après avoir créé la clé USB, il faut monter (importer) et démonter (exporter) la clé correctement pour éviter la cohérence des données iproblèmes. Il faut s'assurer que la clé USB est prête à être retirée en indiquant à ZFS d'écrire toutes les données sur la clé USB. Avant de retirer la clé usb, utiliser “zpool export” et pour monter la clé usb utiliser “zpool import”. Voici des exemples de la “sauvegarde” du pool ZFS que nous avons faite plus tôt.

- brancher la clé usb et monter avec “import”

```
zpool import backup
```

Il faut nettoyer souvent la clé pour assurer la cohérence des données. Au moins chaque fois que l'on veut copier de nouvelles données et avant de démonter la clé USB

`zpool scrub backup`

- Pour démonter la clé USB, utiliser l'export AVANT de débrancher la clé USB !

```
zpool export backup
```

Tout type de clé USB peut être utilisé pour ZFS, mais il existe des clés USB vraiment lentes sur le marché. Il est fortement conseillé d'utiliserune clé USB 3.0 car l'usb 2.0 est en semi-duplex et limité à 35 Mo/sec tandis que l'usb 3.0 est en duplex intégral et limité à 400 Mo/sec. Ainsi, avec l'usb 3.0, on peut lire et écrire en même temps et obtenir des vitesses maximales à partir de la clé USB, tandis que l'usb 2.0 ne peut communiquer que dans un sens à la fois.

1)

GNOP, permet de créer une tranche à partir du disque, et le nouveau périphérique pointera vers les parties de la table GPT

2)

Nombre maximal de secondes entre les groupes de transactions. Le groupe de transactions actuel sera écrit dans le pool et un nouveau groupe de transactions sera lancé si ce laps de temps s'est écoulé depuis le groupe de transactions précédent. Un groupe de transactions peut être déclenché plus tôt si suffisamment de données sont écrites. La valeur par défaut est de 5 secondes. Une valeur plus élevée peut améliorer les performances de lecture en retardant les écritures asynchrones, mais cela peut entraîner des performances inégales lors de l'écriture du groupe de transactions.