Installing Arch Linux on ZFS

This article details the steps required to install Arch Linux onto a ZFS root filesystem.

Contents

- 1 Installation
  - 1.1 Embedding archzfs into archiso
- 2 Partition the destination drive
  - 2.1 Partition scheme
  - 2.2 Example parted commands
- 3 Format the destination disk
- 4 Setup the ZFS filesystem
  - 4.1 Create the root zpool
  - 4.2 Create your datasets
  - 4.3 Configure the root filesystem
- 5 Install and configure Arch Linux
- 6 Install and configure the bootloader
  - 6.1 Using GRUB with BIOS and EFI motherboards

Related articles

ZFS

Experimenting with ZFS

- 6.1.1 error: failed to get canonical path of
- 6.1.2 Booting your kernel and initrd from ZFS
- 6.2 Using rEFInd with UEFI motherboards
- 7 Unmount and restart
- 8 After the first boot
- 9 Native encryption
- 10 See also

Installation

See **ZFS#Installation** for installing the ZFS packages. If installing Arch Linux onto ZFS from the archiso, it would be easier to use the **archzfs** repository.

Embedding archzfs into archiso

See **ZFS** article.

Partition the destination drive

Review **Partitioning** for information on determining the partition table type to use for ZFS. ZFS supports GPT and MBR partition tables.

ZFS manages its own partitions, so only a basic partition table scheme is required. The partition that will contain the ZFS filesystem should be of the type `bf00`, or "Solaris Root".

Drives larger than 2TB require a GPT partition table. GRUB on BIOS/GPT configurations require a small (1~2MiB) BIOS boot partition to embed its image of boot code.

Depending upon your machine's firmware or your choice of boot mode, booting may or may not require an EFI partition. On a BIOS machine (or a UEFI machine booting in legacy mode) EFI partition is not required. Consult **Boot loaders** for more info.

Partition scheme

Here is an example of a basic partition scheme that could be employed for your ZFS root install on a BIOS/MBR installation using GRUB:

```
Part     Size   Type
----     ----   -------------------------
   1     XXXG   Solaris Root (bf00)
```

Using GRUB on a BIOS (or UEFI machine in legacy boot mode) machine but using a GPT partition table:

```
Part     Size   Type
----     ----   -------------------------
   1       2M   BIOS boot partition (ef02)
   2     XXXG   Solaris Root (bf00)
```

Another example, this time using a UEFI-specific bootloader (such as **rEFInd**) and GPT:

```
Part     Size   Type
----     ----   ------------------------
   1     100M   EFI boot partition (ef00)
   2     XXXG   Solaris Root (bf00)
```

ZFS does not support swap files. If you require a swap partition, see **ZFS#Swap volume** for creating a swap ZVOL.

Tip: Bootloaders with support for ZFS are described in **#Install and configure the bootloader**.

Warning: Several GRUB bugs (**bug #42861 (https://savannah.gnu.org/bugs/?42861)**, **zfsonlinux/grub/issues/5 (https://github.com/zfsonlinux/grub/issues/5)**) complicate installing it on ZFS partitions, see **#Install and configure the bootloader** for a workaround

Example parted commands

Here are some example commands to partition a drive for the second scenario above ie using BIOS/legacy boot mode with a GPT partition table and a (slighty more than) 1MB BIOS boot partition for GRUB:

```
# parted /dev/sdx
(parted)mklabel gpt
(parted)mkpart non-fs 0% 2
(parted)mkpart primary 2 100%
(parted)set 1 bios_grub on
```

```
(parted)set 2 boot on
(parted)quit
```

You can achieve the above in a single command like so:

```
parted --script /dev/sdx mklabel gpt mkpart non-fs 0% 2 mkpart primary 2 100% set 1 bios_grub on set 2 boot on
```

If you are creating an EFI partition then that should have the boot flag set instead of the root partition.

Format the destination disk

If you have opted for a boot partition as well as any other non-ZFS system partitions then format them. Do not do anything to the Solaris partition nor to the BIOS boot partition. ZFS will manage the first, and your bootloader the second.

Setup the ZFS filesystem

First, make sure the ZFS modules are loaded,

```
# modprobe zfs
```

Create the root zpool

```
# zpool create -f zroot /dev/disk/by-id/id-to-partition-partx
```

Warning:

- Always use id names when working with ZFS, otherwise import errors will occur.
- The zpool command will normally activate all features. See **ZFS#GRUB-compatible pool creation** when using **GRUB**.

Create your datasets

Instead of using conventional disk partitions, ZFS has the concept of datasets to manage your storage. Unlike disk partitions, datasets have no fixed size and allow for different attributes, such as compression, to be applied per dataset. Normal ZFS datasets are mounted automatically by ZFS whilst legacy datasets are required to be mounted using fstab or with the traditional mount command.

One of the most useful features of ZFS is boot environments. Boot environments allow you to create a bootable snapshot of your system that you can revert to at any time instantly by simply rebooting and booting from that boot environment. This can make doing system updates much safer and is also incredibly useful for developing and testing software. In order to be able to use **beadm (https://github.com/b333z/beadm)** to manage boot environments

your datasets must be configured properly. Key to this are that you split your data directories (such as `/home`) into datasets that are distinct from your system datasets and that you do not place data in the root of the pool as this cannot be moved afterwards.

You should always create a dataset for at least your root filesystem and in nearly all cases you will also want `/home` to be in a separate dataset. You may decide you want your logs to persist over boot environments. If you are a running any software that stores data outside of `/home` (such as is the case for database servers) you should structure your datasets so that the data directories of the software you want to run are separated out from the root dataset.

With these example commands, we will create a basic boot environment compatible configuration comprising of just root and `/home` datasets with lz4 compression to save space and improve IO performance:

```
# zfs create -o mountpoint=none zroot/data
# zfs create -o mountpoint=none zroot/ROOT
# zfs create -o compression=lz4 -o mountpoint=/ zroot/ROOT/default
# zfs create -o compression=lz4 -o mountpoint=/home zroot/data/home
```

Note: You will need to enable ACL support on the pool that will house `/var/log/journal`, i.e. `zfs set acltype=posixacl ...`. See **Systemd#systemd-tmpfiles-setup.service fails to start at boot** for more information.

Configure the root filesystem

If you have just created your zpool, it will be mounted in a dir at the root of your tree named after the pool (ie /zroot). If the following set commands fail, you may need to unmount any ZFS filesystems first:

```
# zfs umount -a
```

Now set the mount points of the datasets:

```
# zfs set mountpoint=/ zroot/ROOT/default
# zfs set mountpoint=legacy zroot/data/home
```

Note: `/etc/fstab` mounts occur before zfs mounts, so don't use zfs mountpoints on directories with subfolders configured to be mounted by `/etc/fstab`.

and put them in `/etc/fstab`

/etc/fstab

```
# <file system>        <dir>         <type>    <options>             <dump> <pass>
zroot/ROOT/default / zfs defaults,noatime 0 0
zroot/data/home /home zfs defaults,noatime 0 0
```

All legacy datasets must be listed in `/etc/fstab` or they will not be mounted at boot.

Set the bootfs property on the descendant root filesystem so the boot loader knows where to find the operating system.

```
# zpool set bootfs=zroot/ROOT/default zroot
```

Export the pool,

```
# zpool export zroot
```

Warning: Do not skip this, otherwise you will be required to use `-f` when importing your pools. This unloads the imported pool.

Note: This might fail if you added a swap partition. You need to turn it off with the *swapoff* command.

Finally, re-import the pool,

```
# zpool import -d /dev/disk/by-id -R /mnt zroot
```

Note: `-d` is not the actual device id, but the `/dev/by-id` directory containing the symbolic links.

If this command fails and you are asked to import your pool via its numeric ID, run `zpool import` to find out the ID of your pool then use a command such as: `zpool import 9876543212345678910 -R /mnt zroot`

If there is an error in this step, you can export the pool to redo the command. The ZFS filesystem is now ready to use.

Be sure to bring the `zpool.cache` file into your new system. This is required later for the ZFS daemon to start.

```
# cp /etc/zfs/zpool.cache /mnt/etc/zfs/zpool.cache
```

if you do not have `/etc/zfs/zpool.cache`, create it:

```
# zpool set cachefile=/etc/zfs/zpool.cache zroot
```

Install and configure Arch Linux

Follow the following steps using the **Installation guide**. It will be noted where special consideration must be taken for ZFSonLinux.

- First mount any legacy or non-ZFS boot or system partitions using the mount command.
- Install the base system.
- The procedure described in **Installation guide#Fstab** is usually overkill for ZFS. ZFS usually auto mounts its own partitions, so we do not need ZFS partitions in `fstab` file, unless the user made legacy datasets of system directories. To generate the `fstab` for

filesystems, use:

```
# genfstab -U -p /mnt >> /mnt/etc/fstab
```

- Edit the `/etc/fstab`:

Note:

- If you chose to create legacy datasets for system directories, keep them in this `fstab`!
- Comment out all non-legacy datasets apart from the root dataset, the swap file and the boot/EFI partition. It is a convention to replace the swap's uuid with `/dev/zvol/zroot/swap`.

- You need to add the **Arch ZFS** repository to `/etc/pacman.conf`, sign its key and **install** **zfs-linux** (or **zfs-linux-lts** if you are running the LTS kernel) within the arch-chroot before you can update the ramdisk with ZFS support.

- When creating the initial ramdisk, first edit `/etc/mkinitcpio.conf` and add `zfs` before filesystems. Also, move `keyboard` hook before `zfs` so you can type in console if something goes wrong. You may also remove fsck (if you are not using Ext3 or Ext4). Your `HOOKS` line should look something like this:

```
HOOKS="base udev autodetect modconf block keyboard zfs filesystems"
```

When using systemd in the initrd, you need to install **mkinitcpio-sd-zfs (https://aur.archlinux.org/packages/mkinitcpio-sd-zfs/)**[AUR] and add the `sd-zfs` hook after the `systemd` hook instead of the `zfs` hook. Keep in mind that this hook uses different kernel parameters than the default `zfs` hook, more information can be found at the **project page (https://github.com/dasJ/sd-zfs)**.

Note:

- If you are using a separate dataset for `/usr` and have followed the instructions below, you must make sure you have the `usr` hook enabled after `zfs`, or your system will not boot.

- **Regenerate the initramfs**.

Install and configure the bootloader

Using GRUB with BIOS and EFI motherboards

Install GRUB onto your disk as instructed here: **GRUB#BIOS systems** or **GRUB#UEFI systems**. The GRUB **manual (https://www.gnu.org/software/grub/manual/grub.html#Configuration)** provides detailed information on manually configuring the software which you can supplement with **GRUB** and **GRUB/Tips and tricks**.

error: failed to get canonical path of

`grub-mkconfig` fails to properly generate entries for systems hosted on ZFS.

```
# grub-mkconfig -o /boot/grub/grub.cfg
/usr/bin/grub-probe: error: failed to get canonical path of `/dev/bus-Your_Disk_ID-part#'
```

```
grub-install: error: failed to get canonical path of `/dev/bus-Your_Disk_ID-part#'
```

To work around this you must set this environment variable: `ZPOOL_VDEV_NAME_PATH=1`. For example:

```
# ZPOOL_VDEV_NAME_PATH=1 grub-mkconfig -o /boot/grub/grub.cfg
```

Booting your kernel and initrd from ZFS

You may skip this section if you have your kernel and initrd on a separate `/boot` partition using something like ext4 or vfat.

Otherwise grub needs to load your kernel and initrd are from a ZFS dataset the kernel and initrd paths have to be in the following format:

```
/dataset/@/actual/path
```

Example with Arch installed on the root dataset:

/boot/grub/grub.cfg

```
set timeout=5
set default=0

menuentry "Arch Linux" {
    search -u UUID
    linux /@/boot/vmlinuz-linux zfs=zroot rw
    initrd /@/boot/initramfs-linux.img
}
```

Example with Arch installed on a nested dataset:

/boot/grub/grub.cfg

```
set timeout=5
set default=0

menuentry "Arch Linux" {
    search -u UUID
    linux /ROOT/default/@/boot/vmlinuz-linux zfs=zroot/ROOT/default rw
    initrd /ROOT/default/@/boot/initramfs-linux.img
}
```

Example with a separate non-ZFS /boot partition and Arch installed on a nested dataset:

/boot/grub/grub.cfg

```
set timeout=5
set default=0

menuentry "Arch Linux" {
```

```
    search -u UUID
    linux /vmlinuz-linux zfs=zroot/ROOT/default rw
    initrd /initramfs-linux.img
}
```

Using rEFInd with UEFI motherboards

Use `EFISTUB` and `rEFInd` for the UEFI boot loader. The kernel parameters in `refind_linux.conf` for ZFS should include `zfs=bootfs` or `zfs=zroot` so the system can boot from ZFS. The `root` and `rootfstype` parameters are not needed.

Unmount and restart

We are almost done!

```
# exit
# umount /mnt/boot (if you have a legacy boot partition)
# zfs umount -a
# zpool export zroot
```

Now reboot.

Warning: If you do not properly export the zpool, the pool will refuse to import in the ramdisk environment and you will be stuck at the busybox terminal.

After the first boot

If everything went fine up to this point, your system will boot. Once. For your system to be able to reboot without issues, you need to enable the `zfs.target` to auto mount the pools and set the hostid.

For each pool you want automatically mounted execute:

```
# zpool set cachefile=/etc/zfs/zpool.cache <pool>
```

Enable the target with **systemd**:

```
# systemctl enable zfs.target
```

When running ZFS on root, the machine's hostid will not be available at the time of mounting the root filesystem. There are two solutions to this. You can either place your spl hostid in the **kernel parameters** in your boot loader. For example, adding `spl.spl_hostid=0x00bab10c`, to get your number use the `hostid` command.

The other, and suggested, solution is to make sure that there is a hostid in `/etc/hostid`, and then regenerate the initramfs image which will copy the hostid into the initramfs image. To write the hostid file safely you need to use the `zgenhostid` command.

To use the libc-generated hostid (recommended):

```
# zgenhostid $(hostid)
```

To use a custom hostid (must be hexadecimal and 8 characters long):

```
# zgenhostid deadbeef
```

To let the tool generate a hostid:

```
# zgenhostid
```

Don't forget to regenerate your image using **mkinitcpio**. Your system should work and reboot properly now.

Native encryption

Warning: Encryption does not exist in a stable release, yet. So do this at you own risk, since it might break.

To use native ZFS encryption, you will need a recent enough zfs package like **zfs-linux-git (https://aur.archlinux.org/packages/zfs-linux-git/)**[AUR] 0.7.0.r26 or newer and embed it into the archiso. Then just follow the normal procedure shown before with the exception that you add the following parameters when creating the dataset:

```
# zfs create -o encryption=on -o keyformat=passphrase -o mountpoint=none zroot/ROOT
# zfs create -o encryption=on -o keyformat=passphrase -o mountpoint=none zroot/data
```

If you want a single passphrase for both your root and home partition, encrypt only one dataset instead:

```
# zfs create -o encryption=on -o keyformat=passphrase -o mountpoint=none zroot/encr
# zfs create -o mountpoint=none zroot/encr/ROOT
# zfs create -o mountpoint=none zroot/encr/data
```

When importing the pool use `-l`, to decrypt all datasets

```
# zpool import -d /dev/disk/by-id -R /mnt -l zroot
```

On reboot, you will be asked for your passphrase.

See also

- **HOWTO install Ubuntu to a Native ZFS Root (https://github.com/dajhorn/pkg-zfs/wiki/HOWTO-install-Ubuntu-to-a-Native-ZFS-Root-Filesystem)**
- **ZFS cheatsheet (http://lildude.co.uk/zfs-cheatsheet)**
- **Funtoo ZFS install guide (http://www.funtoo.org/wiki/ZFS_Install_Guide)**

Retrieved from "https://wiki.archlinux.org/index.php?title=Installing_Arch_Linux_on_ZFS&oldid=506777"

- This page was last edited on 10 January 2018, at 00:10.
- Content is available under GNU Free Documentation License 1.3 or later unless otherwise noted.