NixOS ❄: tmpfs as root

2020-05-02 · 6 minutes read · NixOS Linux Tmpfs Impermanence

This post covers both EFI and legacy boot setups.

One fairly unique property of NixOS is the ability to boot with only `/boot` and `/nix`. Nothing else is actually required. This supports doing all sorts of weird things with your root file system.

One way is to do like Graham's post "erase your darlings" describes and empty your root file system each boot using ZFS snapshots. This way have some cool things that you could do on top of his setup, such as doing snapshots when it's running and roll-back to empty on boot. That way you actually can go back to recover files you lost but still have an empty state.

Another way is to go the tmpfs way which is probably why you're here. So I'm going to walk through the install with tmpfs as root file system.

Step 1 - Partitioning on EFI

I'm going to do the most basic setup when it comes to file systems, just a `/boot` as `fat32` and `/nix` as `ext4` without encryption. If you want to have another file system or use LUKS or something it should be trivial to just format the drive differently and mount it.

```
# Defining a helper variable to make the following
# commands shorter.
DISK=/dev/disk/by-id/ata-VENDOR-ID-OF-THE-DRIVE

# Create partition table
parted $DISK -- mklabel gpt

# Create a /boot as $DISK-part1
parted $DISK -- mkpart ESP fat32 1MiB 512MiB
parted $DISK -- set 1 boot on

# Create a /nix as $DISK-part2
parted $DISK -- mkpart Nix 512MiB 100%
```

Step 1 - Partitioning for legacy boot

I'm going to do the most basic setup when it comes to file systems, just a `/boot` as `ext4` and `/nix` as `ext4` without encryption. If you want to have another file system or use LUKS or something it should be trivial to

just format the drive differently and mount it.

```
# Defining a helper variable to make the following
# commands shorter.
DISK=/dev/disk/by-id/ata-VENDOR-ID-OF-THE-DRIVE

# Create partition table
parted $DISK -- mklabel msdos

# Create a /boot as $DISK-part1
parted $DISK -- mkpart primary ext4 1M 512M
parted $DISK -- set 1 boot on

# Create a /nix as $DISK-part2
parted $DISK -- mkpart primary ext4 512MiB 100%
```

Step 2 - Creating the file systems

This is fairly straight forward in my example:

```
# /boot partition for EFI
mkfs.vfat $DISK-part1

# /boot partition for legacy boot
mkfs.ext4 $DISK-part1

# /nix partition
mkfs.ext4 $DISK-part2
```

Step 3 - Mounting the file systems

Here we do one of the neat tricks, we mount tmpfs instead of a partition and then we mount the partitions we just created.

```
# Mount your root file system
mount -t tmpfs none /mnt

# Create directories
mkdir -p /mnt/{boot,nix,etc/nixos,var/log}

# Mount /boot and /nix
mount $DISK-part1 /mnt/boot
mount $DISK-part2 /mnt/nix

# Create a directory for persistent directories
mkdir -p /mnt/nix/persist/{etc/nixos,var/log}

# Bind mount the persistent configuration / logs
mount -o bind /mnt/nix/persist/etc/nixos /mnt/etc/nixos
mount -o bind /mnt/nix/persist/var/log /mnt/var/log
```

Step 4 - Configuration

Then go ahead and do a `nixos-generate-config --root /mnt` to get a basic configuration for your system.

Step 4.1 - Configure disks

One thing you *want* to do that isn't needed when you install on a normal file system is that you want to set options for your root file system.

So go ahead, open up `hardware-configuration.nix` and add the following options to your root file system.

The most important bit is the `mode`, otherwise certain software (such as openssh) won't be happy with the permissions of the file system.

The `size` is something you can adjust depending on how much garbage you are willing to store in ram until you run out of space on your root. 2G is usually big enough for most of my systems.

```
{
  # …

  fileSystems."/" = {
    device = "none";
    fsType = "tmpfs";
    options = [ "defaults" "size=2G" "mode=755" ];
  };

  # …
}
```

Step 4.2 - Configure users

When you have a system with a tmpfs root you have to configure all users and passwords in `configuration.nix`, otherwise you won't have any user or a password on the next boot.

You probably want to have immutable users as well since it doesn't make any sense to have mutability of users if it's going to reset anyways.

Note: Don't use the options `password` or `hashedPassword` for users because it won't work. It has to be the options named `initialPassword` or `initialHashedPassword`.

```
{
  # …

  # Don't allow mutation of users outside of the config.
  users.mutableUsers = false;

  # Set a root password, consider using initialHashedPassword instead.
  #
  # To generate a hash to put in initialHashedPassword
  # you can do this:
  # $ nix-shell --run 'mkpasswd -m SHA-512 -s' -p mkpasswd
  users.users.root.initialPassword = "hunter2";

  # …
}
```

Make sure to add your own user with a password. The password for the root user is of course optional. But it may be quite useful.

Step 4.3 - Configure persistent files / directories

You probably want to have some more persistent files than the two bind mounts we already have created during the setup.

Adding persistent files from etc:

```
{
  # …

  # machine-id is used by systemd for the journal, if you don't
  # persist this file you won't be able to easily use journalctl to
  # look at journals for previous boots.
  environment.etc."machine-id".source
    = "/nix/persist/etc/machine-id";
```

```
  # if you want to run an openssh daemon, you may want to store the
  # host keys across reboots.
  #
  # For this to work you will need to create the directory yourself:
  # $ mkdir /nix/persist/etc/ssh
  environment.etc."ssh/ssh_host_rsa_key".source
    = "/nix/persist/etc/ssh/ssh_host_rsa_key";
  environment.etc."ssh/ssh_host_rsa_key.pub".source
    = "/nix/persist/etc/ssh/ssh_host_rsa_key.pub";
  environment.etc."ssh/ssh_host_ed25519_key".source
    = "/nix/persist/etc/ssh/ssh_host_ed25519_key";
  environment.etc."ssh/ssh_host_ed25519_key.pub".source
    = "/nix/persist/etc/ssh/ssh_host_ed25519_key.pub";

  # …
}
```

From here you can probably figure out how to do more bind-mounts and symbolic links in `/etc` for the files you want to live across reboots.

A useful tool to discovering files in your tmpfs is *ncdu*, I tend to run `sudo ncdu -x /` to walk around the directory tree to see if there's something I want to make persistent.

You may want to make your `/home` persistent, that can be done by a mount or bind-mount. I have that as tmpfs as well, but that is probably enough content for it's own post.

Update: Now there's a follow-up post for putting tmpfs on `/home` as well, it's located here: NixOS ❄: tmpfs as home.

Step 4.4 - Configure the boot loader

Here you can choose your boot loader.

```
{
  # …

  # Use systemd boot (EFI only)
  boot.loader.systemd-boot.enable = true;
  boot.loader.efi.canTouchEfiVariables = true;

  # Use the GRUB 2 boot loader (Both EFI and legacy boot supported).
  boot.loader.grub.enable = true;

  # This is for GRUB in EFI mode
  boot.loader.grub.efiSupport = true;
  boot.loader.grub.device = "nodev";

  # This is for GRUB for legacy boot
  boot.loader.grub.version = 2;
  boot.loader.grub.device = "/dev/sda";

  # …
}
```

Step 4.5 - Configure the rest of your system

You should make sure to go through `configuration.nix` to make all necessary configuration you want such as boot loader, `hostname`, `timezone`, `keymap`, personal user, network settings, etc.

Step 5 - Perform install

Perform the actual install. We can ignore setting the roots password at the end of the install since it won't be there after reboot anyway.

```
nixos-install --no-root-passwd
```


Made with in Arvika by Taserud Consulting AB.