Back to Ubuntu: Linux dual-boot, but cheaper

Posted on October 21, 2022 by Linus Heckemann

I love NixOS. But sometimes, proprietary applications are too fussy about their environment, don’t work in an FHS user env because they want to do their own fun stuff with namespacing, and on top of that want direct access to hardware, set up their own systemd services, or otherwise cause trouble.

I was faced with such a scenario, so I thought I should give the application a try in an Ubuntu installation. It turns out to be surprisingly easy to set up a “lite dual-boot” environment which allows doing this with minimal changes to the NixOS system.

Background: the boot process

Nowadays, most distributions that support a wide range of hardware will boot using an *initramfs*, a filesystem that is loaded by the bootloader and contains the tools to perform some initialisation that the kernel doesn’t necessarily know how to do on its own, before switching into the main operating system tree. A typical initramfs init script will:

- Load drivers for boot-critical hardware such as disk controllers; this is often done automatically by running udev on modern systems, though some drivers may still need to be loaded explicitly using `modprobe`.
- Run software utilities needed to access the root filesystem; this may mean setting up LVM, importing a ZFS pool, opening encrypted volumes, or configuring the network if the root filesystem is accessed via the network.
- Mount the operating system’s root filesystem.

Finally, it uses the `pivot_root` system call to switch the new root filesystem in, and move the initramfs filesystem to a different location. It can then execute the real system’s init script or executable.

The principle of this installation method is to use `pivot_root` from a fully booted NixOS system in order to switch into an Ubuntu filesystem. The result is a booted Ubuntu system, which differs from a normally installed Ubuntu only in a few aspects:

- It boots using the NixOS kernel, not an Ubuntu-provided one;
- It doesn’t know anything about the filesystems, these are entirely initialised by NixOS and not touched by Ubuntu.

Implementation details

First, we set up a root directory for the Ubuntu installation. I created a zfs dataset, which allows me to take snapshots of the Ubuntu installation and roll them back and such:

```
[root@ordnungsamd:~]# zfs create -o mountpoint=/ubuntu ordnungsamd/ubuntu
```

The Ubuntu root directory needs to be a mountpoint for `switch_root` to work, but if your filesystem of choice doesn’t support subvolumes or similar you can use a bind mount to satisfy this requirement:

```
[root@ext4pc:~]# mkdir /ubuntu
[root@ext4pc:~]# mount --bind /ubuntu /ubuntu
```

We then set up the ubuntu root filesystem using `debootstrap`, the Debian installer:

```
[root@ordnungsamd:~]# nix-shell -p debootstrap --run 'debootstrap focal /ubuntu/ ht
```

It will print the following warning:

```
W: Cannot check Release signature; keyring file not available /usr/share/keyrings/u
```

I’ve spent quite some time trying to find a place where I can simply download the keys in question; unfortunately, I haven’t found a good (official Ubuntu, HTTPS, easy access) source for the keyring! The only way I’ve come up with at the moment is downloading an Ubuntu ISO via HTTPS, then extracting it from there. That’s a lot of overhead. If anyone knows where I might be able to download this keyring from ubuntu via HTTPS, I’d be glad to hear it.

Since debootstrap doesn’t clean up nicely after itself, we need to unmount the filesystems it created and take out the resolv.conf copied in from the host system (systemd-nspawn will create this on the fly for the container stage, and Ubuntu’s own networking setup will create this when booting fully):

```
[root@ordnungsamd:~]# umount /ubuntu/{proc,sys}
[root@ordnungsamd:~]# rm /ubuntu/etc/resolv.conf
```

At this point we could already boot into it. However, we need a few more bits and pieces to make it properly usable. We enter the system like a container in order to set these up, and source `/etc/environment` to set the PATH environment variable according to Ubuntu’s expectations:

```
[root@ordnungsamd:~]# systemd-nspawn -D /ubuntu
Spawning container ubuntu on /ubuntu.
Press ^] three times within 1s to kill container.
-bash: groups: command not found
-bash: lesspipe: command not found
-bash: dircolors: command not found
root@ubuntu:~# . /etc/environment
```

Our missing pieces are: - a user to log in as; I ran `adduser linus` and filled out the prompts.

- a root password: run `passwd` to set this in order to be able to become root within the booted system, though this isn’t even strictly necessary if we get everything right at this point!

- a desktop: run `apt install ubuntu-desktop-minimal` to install only the bits we really need. For a fully-fledged Ubuntu desktop installation, one can also use `ubuntu-desktop`. If this is only for non-graphical applications, this is also not necessary, but an SSH server and networking setup would be good to have.

Once all this is set up, we can tell NixOS’s systemd to shut down the NixOS system and pass on to Ubuntu’s, using `systemctl switch-root /ubuntu /sbin/init`. This will shut down the system as would normally happen when powering off or rebooting, but instead of powering the system off or triggering a reset, systemd will `pivot_root` into `/ubuntu` and then execute `/sbin/init`, Ubuntu’s own init.

I usually return to the NixOS system by rebooting as normal, but mounting the NixOS filesystems then using `systemctl switch-root` to switch back should work too.

Summary

Overall, this process took me about half an hour on my first attempt, which I think is similar to the amount of time it would have taken me to download an Ubuntu CD image, write it to a USB stick, and reinstall — not to mention that this would have been a lot trickier while keeping my daily-driver NixOS installation functional! It served me well for the intended purposes for over 2 years, until the game I was playing started working on NixOS :)

To sum up, a comparison of this setup with a typical Ubuntu installation:

Advantages

- Fast installation
- No repartitioning necessary
- Easy to remove
- Low risk: does not touch existing boot process

Disadvantages

- No Ubuntu kernel: this approach is nearly useless for comparing hardware issues or running software that requires its own kernel modules (like VirtualBox). It also means that hotplugging hardware for which the drivers weren’t already loaded will not work.
- Unsupported: this setup is quite different in nature from a normal Ubuntu install, and asking for help with it is likely to result in people unwilling to help you because your setup is all weird!
