Nix-on-Droid

when the low-hanging fruit is also the ripest one

Alexander Sosedkin (@t184256)

recently Red Hat, formerly a plasma physicist

2019-10-26

Running your favourite userland on mobile devices

- Get rid of Android and port your distro
- Recompile it with Bionic (Android's special libc)
- Get root and do what it takes (usually a chroot)
- Run it under userspace chroot emulation

Boring details

Nix-on-Droid

It's Nix, running on Android, under `proot`:
userspace bind-mounting emulation implemented through `ptrace`.

This was done to avoid recompiling nixpkgs and reusing aarch64 packages from Hydra.

It also comes with an terminal emulator app that handles the bootstrapping process.

A few more hacks are also in-place because Bionic is different.

Decomposition

Nix-on-Droid is made up of:

- a cross-compiled Bionic `proot`,
- an Android terminal emulator app,
- Nix expressions for a boostrap zipball and
- a Nix channel.

proot

Intercepts file-system access to rewrite `/nix/store` to `/data/data/com.termux.nix/files/usr/nix`. Also rewrites access to `/etc/` to make glibc happy while we're on it.

Also, emulates hardlinks with symlinks.

Also, effectively breaks other `ptrace` users.

Also, makes everything slower and buggier (sigh).

But it makes it superportable: no root, no user namespaces.

App

It is a fork of Termux-the-terminal-emulator app, which downloads a bootstrap zipball from https://nix-on-droid.unboiled.info, unpacks it, sets up symlinks and runs the login script, which, it turns, runs `proot` and, finally, drops you into a Nix-powered shell.

On subsequent launches, it also has to resolve a maze of broken symlinks that points to the login script, which is now managed by Nix.

Has a ‘failsafe’ functionality that drops you into an Android shell that one could use in case everything went up in flames.

Bootstrap zipball

It's a mess.

It starts with cross-compiling an Android-geared fork of `proot` with Android NDK targeting Bionic.

Then it takes an official Nix release tarball, which undergoes minor modifications, performs some light initialization through QEMU's user mode emulation and gets repacked into an esoteric format invented for Termux.

Then you put it to some HTTP server so that the app can download it on the first startup.

It used to be a giant shell script, but @Gerschtli rewrote it all and now it is built with Nix.

Channel and update path

Finally (again, thanks to @Gerschtli) nix-on-droid-bootstrap repository doubles as a Nix channel, so that the scripts that are used to ‘dive’ into the Nix-powered environment are also managed by Nix itself, and the user has an actual upgrade path.

Currently, the outermost Bionic `proot` binary is the last thing that’s not under control of Nix, but this is planned to change.

Limitations

- You’ve installed Nix from a regular user
- under Android, with no root and weird everything.
- Proot complicates and slows down things.
- Currently only aarch64 and (untested) i686.

Story

A quest for a lighter laptop

This spring I wanted a lighter travel laptop,
and ended up eyeing up an awesome Android tablet.

Why not a tablet? I don't need much from it, but… could it run Nix?

ghost commented on May 13, 2017

This would turn toys into semi-computers. Should just entirely replace apt with nix.

4

If I get a tablet, I want it to stay a tablet

Even if it could run nixos-mobile, I'll miss Android.
Porting aside, mobile UX is years behind.

So, what I wanted was Nix on Android.

I've also just read @matthewbauer's posts on statically linking Nix and cross-compilation and I felt ready.


I HAVE NO IDEA WHAT I'M DOING

It just worked (thanks to you)!

I bought that tablet and I had it working the first evening.
I didn't even get to compile anything!

The hardest part was to understand that I don't need to do anything. Nix release tarball already ships with everything you need to run Nix.

I got `proot` to pretend that the store is at `/nix/store` and fake a couple of files in `/etc`. I downloaded and unpacked the official Nix tarball.

Boom, Nix working on Android. Playing by Android rules and limitations.

Evolution

Over time, the thing evolved a lot:

- I've cut out the middle man and forked the terminal app to run Nix
- I wrote a Nix expression for cross-compiling `proot` (thanks, @matthewbauer).
- I've reached 'just works for me' stage.
- @Gerschtli rewrote the building with Nix.
- @Gerschtli put my hacks under Nix control and blessed the project with an upgrade path.
- It was ju-ust accepted into F-Droid.

Possibilities / limitations

- You can install the app, open a terminal and get a shell with Nix.
- You are encouraged to install home-manager.
- You can install software from nixpkgs / binary cache.
- You can reuse your configs.
- No root, user namespaces or SELinux disabling required.

- You need aarch64 (or maybe i686).
- It's 3-10 times slower than my i7 laptop.
- You're running everything as an unprivileged user…
- … under Android, an exotic distro. Cooperation is lacking.

What to do with that?

I have `mosh`, `neovim`, `git` and `xonsh` on a 270 g tablet.

I've typeset this presentation (`pandoc+xelatex`) on it.

And all Android experience is still there, intact.

What to do with that?

I have `mosh`, `neovim`, `git` and `xonsh` on a 270 g tablet.

I've typeset this presentation (`pandoc+xelatex`) on it.

And all Android experience is still there, intact.

But I think there's a bigger question.

We have a quirky, but uniform target platform in our pockets.
Think `nix-darwin`, but with more compatible devices in the wild.

I don't know why don't we harvest this fruit.

Questions

Mine:

- Why don’t we target Android if it’s so easy?
- Could we make my project a tad more official?
- Could I recompile everything, but get out of proot?
- What could be the endgame and is it worth the effort?

I’m eager to hear what you think.

Thank you