NixOS Android Builder User Guide

Contents

Requirements 2

Preparations on your local machine 3

- Create Secure Boot Keys 3
- Build an Image 3
- Sign the Image 4
- Flash the Image 5

On the Target Machine 6

- Fetch Android 6
- Build Android 7
- SBOM Generation 8
- Save Build Outputs 8
- Reset to Initial State 8

Updates 9

Automated Tests 9

Customization 10

- nixosAndroidBuilder.build.lunchTarget 10
- nixosAndroidBuilder.build.repoBranch 10
- nixosAndroidBuilder.build.repoManifestUrl 11
- nixosAndroidBuilder.build.sourceDir 11
- nixosAndroidBuilder.build.userEmail 11
- nixosAndroidBuilder.build.userName 11
- nixosAndroidBuilder.debug 11
- nixosAndroidBuilder.fhsEnv.packages 11
- nixosAndroidBuilder.fhsEnv.pins 11

This document describes the steps needed to build a disk image for NixOS Android Builder, boot it on a target machine and build Android Open Source Project with it.

At the end, it provides an overview on customization options as well as guidance on keeping the project up to date with upstream sources.

A more detailed architecture and technical description is available in docs.pdf.

Requirements

- A **local machine** to build the image. Any machine with nix installed, that can build for `x86_64-linux` targets should suffice. It needs…
  - ~4GB of memory for nix evaluation
  - ~30GB of disk space for dependencies as well as the built images
- A **USB mass storage** device, such as an USB stick, to transfer the built image to the target machine. It needs at least 8GB of storage to boot, and at least 260GB if the target machine has no other disk.
- A **target machine** on which to build Android on. It needs:
  - ~64G of memory
  - ~250G of disk space
  - EFI boot with secure boot in setup mode
- A **git checkout** of this repository, containing nix expressions and helper scripts to build the image. Commands in this document assume to be run from the repository's top-level directory, unless explicitly stated otherwise.

Preparations on your local machine

Create Secure Boot Keys

We ship a small helper script, `create-signing-keys`, in our repo that generates a key pair to sign our Unified Kernel Image (`UKI`) with for secure boot. The script is intended to be run from the repository's top-level directory and will write keys, certificates, etc. to a `keys/` directory relative to you current working directory.

Those keys are required for Secure Boot signing and enrollment of target machines. They need to be stored **secure** and **safely**, and **must not** be committed to the repository plain text.

How to handle that is currently left to the user to decide. Various 3rd-party tools such as SOPS can be used for this purpose.

Let's start by running the following command in a local git checkout of this repository:

```
$ nix run .#create-signing-keys
```

```
>>> Generating UUID
56521b4e-c625-4792-9718-07196c355f66

>>> Generating PK
[...]
>>> Generating KEK
[...]
>>> Generating DB
[...]

total 27
drwxr-xr-x  2 user users   19 Sep 19 13:22 .
drwxr-xr-x 10 user users   20 Sep 19 13:22 ..
-rw-------  1 user users 3374 Sep 19 13:22 db.auth
-rw-r--r--  1 user users 1317 Sep 19 13:22 db.cer
-rw-r--r--  1 user users 1838 Sep 19 13:22 db.crt
-rw-r--r--  1 user users 1361 Sep 19 13:22 db.esl
-rw-------  1 user users 3268 Sep 19 13:22 db.key
-rw-r--r--  1 user users   37 Sep 19 13:22 guid.txt
-rw-------  1 user users 3374 Sep 19 13:22 KEK.auth
-rw-r--r--  1 user users 1317 Sep 19 13:22 KEK.cer
-rw-r--r--  1 user users 1838 Sep 19 13:22 KEK.crt
-rw-r--r--  1 user users 1361 Sep 19 13:22 KEK.esl
-rw-------  1 user users 3272 Sep 19 13:22 KEK.key
-rw-------  1 user users 3374 Sep 19 13:22 PK.auth
-rw-r--r--  1 user users 1317 Sep 19 13:22 PK.cer
-rw-r--r--  1 user users 1838 Sep 19 13:22 PK.crt
-rw-r--r--  1 user users 1361 Sep 19 13:22 PK.esl
-rw-------  1 user users 3272 Sep 19 13:22 PK.key
-rw-------  1 user users 2013 Sep 19 13:22 rm_PK.auth
```

The files listed at the end of the output are the generated keys, certificates, etc in `keys`. Our signing script, `sign-disk-images` will pick them up a few steps further down.

But first…

Build an Image

Let's build a ready-to-boot disk image from the nix expressions in this repository.

The following command will automatically download dependencies from the binary cache at cache.nixos.org and build those not already cached. Depending on your internet uplink and the speed of your local machine, the first run may take a long time, but subsequent runs will be much faster due to local caching of artifacts.

Note: A 3rd-party tool such as nix-output-monitor can help to gain a better overview of what's currently being downloaded or built. `nix shell nixpkgs#nom` starts a shell with it, where you can just replace `nix build` with `nom build` below.

```
$ nix build --print-build-logs .#image
```

```
[...]
android-builder> Running phase: installPhase
android-builder> renamed 'repart-output.json' -> 'repart-output_orig.json'
android-builder> removed 'repart-output_orig.json'
android-builder> copied 'repart-output.json' ->
 ↪  '/nix/store/[hash]-android-builder-25.11pre-git/repart-output.json'
android-builder> removed 'repart-output.json'
android-builder> copied 'android-builder_25.11pre-git.raw' ->
 ↪  '/nix/store/[hash]-android-builder-25.11pre-git/android-builder_25.11pre-git.raw'
android-builder> removed 'android-builder_25.11pre-git.raw'
```

When `nix` finishes building the image, it creates a symlink - `result` - that points to a `/nix/store` path containing a raw disk image.

That image already contains a full partition table and our NixOS closure including all tools to build Android. It's almost ready to boot, but not yet signed for secure boot. We are going to fix that in the next step!

We can see the full path as well as the size of the generated disk image by running i.e.:

```
$ du --human-readable --apparent-size "$(realpath result/*.raw)"
```

```
7.9G    /nix/store/[hash]-android-builder-25.11pre-git/android-builder_25.11pre-git.raw
```

(Your disk image's size may be slightly different than this example)

Sign the Image

With our disk image built, we still need to sign it for secure boot, as it still contains an unsigned `UKI` on its EFI System Partition (`ESP`).

`sign-disk-image` signs that `UKI` with the keys generated earlier and copies secure boot update bundles into the image so that the firmware can enroll them automatically.

`sign-disk-image` deliberately runs outside the nix sandbox in order to be able to access our keys in `keys/` without copying them to the world-readable `/nix/store`. So the script starts by copying the image out of the nix store to a temporary file, before signing the `UKI`, copying secure boot update bundles, and - finally moving the image to the repository's top-level directory.

To sign your image, run:

```
$ nix run .#sign-disk-image ./results/*.raw
```

```
Using keystore /home/user/src/android-build-vm/keys.
Copying result/android-builder_25.11pre-git.raw to /tmp/tmp.2Aie6kmiwfandroid-builder.raw
Searching ESP partition offset in /tmp/tmp.2Aie6kmiwfandroid-builder.raw
Copying EFI/BOOT/BOOTX64.EFI from the image to /tmp/tmp.qF9j1K2BOAefi/EFI/BOOT/BOOTX64.EFI
Signing /tmp/tmp.qF9j1K2BOAefi/EFI/BOOT/BOOTX64.EFI
Signing Unsigned original image
Copying /tmp/tmp.qF9j1K2BOAefi/EFI/BOOT/BOOTX64.EFI back to the image, into
 ↪  EFI/BOOT/BOOTX64.EFI
Copying certificates from /home/user/src/android-build-vm/keys to the image, into EFI/KEYS
Moving the image from /tmp/tmp.2Aie6kmiwfandroid-builder.raw to
 ↪  /home/user/src/android-build-vm/android-builder_25.11pre-git.raw
Done. You can now flash the signed image:
/home/user/src/android-build-vm/android-builder_25.11pre-git.raw
```

After this step, a `.raw` disk image with the same name `android-builder_25.11pre-git.raw` should show up in the repository's top-level directory.

Ready to be flashed to a block device in the next step!

Flash the Image

With our image both built and signed, we are now ready to flash it to a block device, such as a USB stick or external hard drive.

Attach the block device to your local machine and make sure you find its *device path*, e.g. via `lsblk`. We will use `/dev/sdX` as the block device, `android-builder_25.11pre-git.raw` as our image name in this example. While any tool to write a disk image to a block device could do the job, we simply use `dd` here and ensure that the kernel's page cache is flushed by waiting for `sync` to finish before detaching the block device from the local machine:

```
$ sudo dd \
  bs=1M status=progress \
  if=android-builder_25.11pre-git.raw \
  of=/dev/sdX
$ sudo sync
```

```
8470429696 bytes (8.5 GB, 7.9 GiB) copied, 14.9754 s, 566 MB/s
```

Replace `/dev/sdX` with the path to your removable device (e.g. `/dev/disk/by-id/...`).

After the `sync` is finished, you can safely move the USB stick or external hard drive to your target machine.

Be sure to enable secure boot setup mode before booting the target machine from the block device for the first time. This is needed to automatically enroll our keys on first boot of the machine.

The enrolled keys stay valid until either the target machines firmware is reset or the keys are rotated. So you do **not** need to put secure boot in setup mode again if i.e. an earlier version of the image already enrolled the same keys.

On the Target Machine

After booting, you will be dropped into a NixOS environment with an interactive shell that contains the Android build tools by default.

Fetch Android

The `fetch-android` script clones the latest AOSP repository into the builder's workspace and checks out the default branch. Run it from the shell:

```
$ fetch-android
```

```
Fetching android:
  repo.branch      = android-latest-release
  repo.manifestUrl = https://android.googlesource.com/platform/manifest

Downloading Repo source from https://gerrit.googlesource.com/git-repo
remote: Sending approximately 12.31 MiB ...
remote: Counting objects: 271, done
remote: Total 9417 (delta 4518), reused 9417 (delta 4518)
repo: Updating release signing keys to keyset ver 2.3
Updating files: 100% (2/2), done.

Your identity is: CI User <ci@example.com>
If you want to change this, please re-run 'repo init' with --config-name

repo has been initialized in /var/lib/build/source
Syncing: 100% (987/987), done in 35m53.176s
repo sync has finished successfully.
Syncing: 100% (987/987), done in 1m53.939s
repo sync has finished successfully.
```

The downloaded source ends up in `/var/lib/build/source` by default. The repository and branch, the git identity as well the as the source directory can be customized, both at run-time and at build-time. See below for build-time customization, or `fetch-android --help` for available run-time flags:

```
$ fetch-android --help
```

```
Usage: /run/current-system/sw/bin/fetch-android [options] [-- ...repo sync args...]

Options:
  --user-email=EMAIL        Git user.email (default: ci@example.com)
  --user-name=NAME          Git user.name (default: CI User)
  --repo-branch=BRANCH      Repo branch to init (default: android-latest-release)
  --repo-manifest-url=URL   Repo manifest URL (default:
↪  https://android.googlesource.com/platform/manifest)
  --source-dir=DIR          Source directory (default: /var/lib/build/source)
  -h, --help                Show this help message
```

Lower-level tools such as `repo`, `git`, `gpg`, etc. are available in the shell as well in case `fetch-android` does not fit your needs.

Build Android

Once the source is available, build your target product with `build-android`. It will automatically source the build environment and call the appropriate Android make command.

```
$ build-android
```

```
Building android:
  lunch.target = aosp_cf_x86_64_only_phone-aosp_current-eng
  make.args    =


============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=16
TARGET_PRODUCT=aosp_cf_x86_64_only_phone
TARGET_BUILD_VARIANT=eng
TARGET_ARCH=x86_64
TARGET_ARCH_VARIANT=silvermont
HOST_OS=linux
HOST_OS_EXTRA=Linux-6.12.44-x86_64-NixOS-25.11-(Xantusia)
HOST_CROSS_OS=windows
BUILD_ID=BP2A.250605.031.A2
OUT_DIR=out
SOONG_ONLY=false
============================================
[...]
[ 38% 5/13 20s remaining] cp if changed
↪  out/soong/.intermediates/build/release/build_config
↪  /build_flag_system_ext/android_common/build_flags.json

#### build completed successfully (12 seconds) ####
```

(Example output is from a run with all artifacts already built. First run will be much more verbose).

The product and variant to build can be customized, both at run-time and at build-time. See below for build-time customization, or `build-android --help` for available run-time flags:

```
$ build-android --help
```

```
Usage: /run/current-system/sw/bin/build-android [options] [-- ...m args...]

Options:
  --source-dir=DIR      Source directory (default: /var/lib/build/source)
  --lunch-target=VALUE  Lunch target (default: aosp_cf_x86_64_only_phone-aosp_current-eng)
  -h, --help            Show this help message
```

Lower-level tools such as `lunch`, `m`, `ninja`, etc. are available in the shell after loading Androids `envsetup.sh`:

```
$ cd /var/lib/build/source
$ source build/envsetup.sh
```

Please refer to upstream documentation for details about using them.

SBOM Generation

`sbom-android` is a lightweight wrapper around `build-android` to generate Software Bills Of Materials (`SBOMs`) using upstreams SBOM facilities.

The resulting `SBOM` can be found in `/var/lib/build/source/out/soong/sbom`.

```
$ sbom-android
```

```
Building android:
  lunch.target = aosp_cf_x86_64_only_phone-aosp_current-eng
  make.args    = sbom


============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=16
TARGET_PRODUCT=aosp_cf_x86_64_only_phone
TARGET_BUILD_VARIANT=eng
TARGET_ARCH=x86_64
TARGET_ARCH_VARIANT=silvermont
HOST_OS=linux
HOST_OS_EXTRA=Linux-6.12.44-x86_64-NixOS-25.11-(Xantusia)
HOST_CROSS_OS=windows
BUILD_ID=BP2A.250605.031.A2
OUT_DIR=out
SOONG_ONLY=false
============================================
[...]
[ 50% 5/10 1m22s remaining] cp if changed
 ↪  out/soong/.intermediates/build/release/build_config
 ↪  /build_flag_product/android_common/build_flags.json

#### build completed successfully (13 seconds) ####
```

(Example output is from a run with all artifacts already built. First run will be much more verbose).

Save Build Outputs

All build outputs, logs and images can be found in `/var/lib/build/source/out` once the build has finished. Images in particular are in `out/target/product`, logs in `error.log` and `verbose.log.gz`.

Since the outputs are still stored on the ephemeral `/var/lib` partition at the moment, they will effectively be deleted when the target machine shuts down for any reason. Copying them to a remote or local persistent storage medium is left to the user at this time.

Reset to Initial State

To start a fresh build, simply reboot.

Shutting down the machine, even sudden power loss, will flush the disk encryption key from RAM and render all previous build artifacts inaccessible. Upon reboot, the `/var/lib` partition will be re-formatted and re-encrypted.

To reboot, use your machine's physical power buttons or run the following in a shell:

```
$ systemctl reboot
```

Updates

All dependencies besides Android sources are coming from `nixpkgs`. So just updating the pinned `nixpkgs` revision in our flake, updates all software packages used to build the image.

To update all flake's inputs, run:

```
$ nix flake update
```

After updating, (re-)builds of the image will use the latest packages from the tracked `nixpkgs` branch - currently the rolling-release branch `nixos-unstable`.

To catch possible regression bugs early, we do recommend to run at least the `integration` test as described in Automated Tests below.

Automated Tests

The repository ships a NixOS VM test that boots a built image and checks the build environment, secure-boot enrollment, and dm-verity status.

To run the tests in a virtual machine on your local machine:

```
$ nix build -L .#checks.x86_64-linux.integration
```

```
Machine state will be reset. To keep it, pass --keep-vm-state
[...]
Copying /nix/store/[hash]-android-builder-25.11test/android-builder_25.11test.raw to
 ↪  android-builder_25.11test.raw
Resizing android-builder_25.11test.raw to 30720M
Image resized.
Signing UKI in android-builder_25.11test.raw
using keystore /nix/store/[hash]-test-keys.
Searching ESP partition offset in android-builder_25.11test.raw
Copying EFI/BOOT/BOOTX64.EFI from the image to
 ↪  /build/tmp.bHZYjbI7BGefi/EFI/BOOT/BOOTX64.EFI
Signing /build/tmp.bHZYjbI7BGefi/EFI/BOOT/BOOTX64.EFI
Signing Unsigned original image
Copying /build/tmp.bHZYjbI7BGefi/EFI/BOOT/BOOTX64.EFI back to the image, into
 ↪  EFI/BOOT/BOOTX64.EFI
Copying certificates from /nix/store/[hash]-test-keys to the image, into EFI/KEYS
Done. You can now flash the signed image:
android-builder_25.11test.raw
android-builder: starting vm
android-builder: QEMU running (pid 37)
android-builder: waiting for unit default.target
android-builder: waiting for the VM to finish booting
[...]
android-builder: (finished: waiting for the VM to finish booting, in 22.43 seconds)
[...]
android-builder: (finished: waiting for unit default.target, in 23.48 seconds)
subtest: secure boot works
android-builder: must succeed: bootctl status
[...]
(finished: subtest: secure boot works, in 0.07 seconds)
subtest: dm-verity works
android-builder: must succeed: veritysetup status usr
android-builder: (finished: must succeed: veritysetup status usr, in 0.04 seconds)
(finished: subtest: dm-verity works, in 0.04 seconds)
subtest: Checking FHS Environment
subtest: /usr/bin/env can be executed
android-builder: must succeed: /usr/bin/env --version
android-builder: (finished: must succeed: /usr/bin/env --version, in 0.02 seconds)
```

```
(finished: subtest: /usr/bin/env can be executed, in 0.02 seconds)
subtest: Executables in /bin can be run
android-builder: must succeed: /bin/diff -v
android-builder: (finished: must succeed: /bin/diff -v, in 0.01 seconds)
(finished: subtest: Executables in /bin can be run, in 0.01 seconds)
subtest: /bin/bash sets default $PATH and is a regular file with the correct linker
android-builder: must succeed: env -i /bin/bash -c 'echo $PATH'
android-builder: (finished: must succeed: env -i /bin/bash -c 'echo $PATH', in 0.01
 ↪  seconds)
android-builder: must succeed: file /bin/bash
android-builder: (finished: must succeed: file /bin/bash, in 0.03 seconds)
(finished: subtest: /bin/bash sets default $PATH and is a regular file with the correct
 ↪  linker, in 0.04 seconds)
subtest: dynamic linkers exist as regular files in /lib(64) and search /lib
android-builder: must succeed: file /lib/ld-linux-x86-64.so.2
android-builder: (finished: must succeed: file /lib/ld-linux-x86-64.so.2, in 0.01 seconds)
android-builder: must succeed: file /lib64/ld-linux-x86-64.so.2
android-builder: (finished: must succeed: file /lib64/ld-linux-x86-64.so.2, in 0.01
 ↪  seconds)
android-builder: must succeed: /lib/ld-linux-x86-64.so.2 --help
android-builder: (finished: must succeed: /lib/ld-linux-x86-64.so.2 --help, in 0.00
 ↪  seconds)
(finished: subtest: dynamic linkers exist as regular files in /lib(64) and search /lib, in
 ↪  0.03 seconds)
(finished: subtest: Checking FHS Environment, in 0.09 seconds)
android-builder: waiting for the VM to power off
[...]
test script finished in 30.40s
cleanup
kill vlan (pid 7)
(finished: cleanup, in 0.00 seconds)
```

Note that the test are only run again if inputs did change since the last run.

Customization

The following NixOS Options are available to customize features & behaviour of NixOS Android Builder at build-time. Some, e.g. those in the `nixosAndroidBuilder.build` namespace, also have equivalent flags to use at run-time where an interactive shell is available.

Settings can be set in `configuration.nix`.

nixosAndroidBuilder.build.lunchTarget

Type: `string`

Default: `"aosp_cf_x86_64_only_phone-aosp_current-eng"`

Name of lunch target to build. Can be overriden at run-time by passing '--lunch-target' to 'build-android'.

nixosAndroidBuilder.build.repoBranch

Type: `string`

Default: `"android-latest-release"`

Name of the branch that should be fetched. Can be overriden at run-time by passing '--repo-branch' to 'fetch-android'.

nixosAndroidBuilder.build.repoManifestUrl

Type: `string`

Default: `"https://android.googlesource.com/platform/manifest"`

URL of a ‘repo‘ manifest to fetch sources from. Can be overriden at run-time by passing ‘--repo-manifest-url‘ to ‘fetch-android‘.

nixosAndroidBuilder.build.sourceDir

Type: `absolute path`

Default: `"/var/lib/build/source"`

Directory where ‘repo‘ checkout is stored. Can be overriden at run-time by passing ‘--source-dir‘ to ‘fetch-android‘ and ‘build-android‘.

nixosAndroidBuilder.build.userEmail

Type: `string`

Default: `"ci@example.com"`

Email address used for ‘repo‘/‘git‘ operations. Can be overriden at run-time by passing ‘--user-email‘ to ‘fetch-android‘.

nixosAndroidBuilder.build.userName

Type: `string`

Default: `"CI User"`

User name used for ‘repo‘/‘git‘ operations. Can be overriden at run-time by passing ‘--user-name‘ to ‘fetch-android‘.

nixosAndroidBuilder.debug

Type: `boolean`

Default: `false`

Whether to enable image customizations for interactive access during run-time.

nixosAndroidBuilder.fhsEnv.packages

Type: `list of package`

Packages needed to build Android AOSP. This is mostly copied from AOSP documentation, but could probably be reduced further, as AOSP repos ship much of it in-tree (i.e. python3, jdk, etc)

nixosAndroidBuilder.fhsEnv.pins

Type: `list of package`

A sorted list of packages to add first, so that they ”win” if there are collisions/conflicts during creation of the FHS env. Unresolved collisions will produce a warning in the build log.