# Cuttlefish and Kernels

Linaro Connect, May 2024
Ram Muthiah <rammuthiah@google.com>

# What is Cuttlefish?

- Android Virtual Device used by kernel, systems, and BSP devs across the Android Ecosystem to help develop pre-silicon hardware, kernel software, or test various different android configurations

# Why should you use it?

- Virtio compliant
  - GPU, SND, Input, Net, Wifi, Block, pmem
  - QEMU, CrosVM, Gem5, QNX, OpenSynergy
- ADB, WebRTC, serial
- Used to test upstream Linux
  - Android Common Kernel’s CI/CD pipeline
- AArch64, x86_64, riscv64
  - GCE, AWS, w/ or w/o GPU, Ampere Boxes, Rockpi, Emulation
- Bootloader support (U-Boot)
  - UEFI compatibility
  - Bootconfig + AVB support
- Fastboot
- Developed upstream (AOSP)

# Getting Started

## Install our host packages

- cuttlefish-base and cuttlefish-user - https://github.com/google/android-cuttlefish

## Android Build

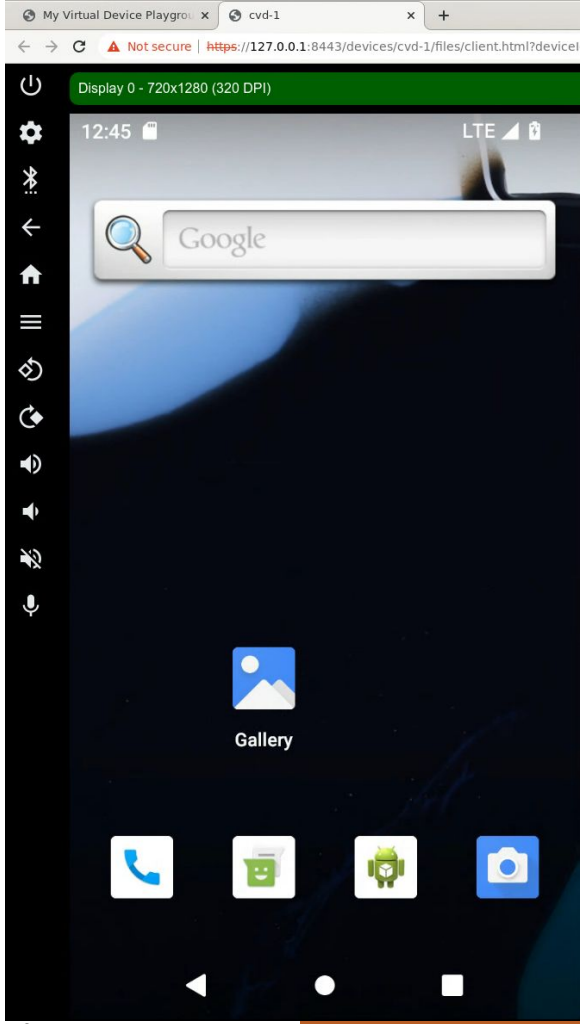
```
$ mkdir android && cd android
$ repo init -u https://android.googlesource.com/platform/manifest -b main
$ repo sync -j
$ source build/envsetup.sh
$ lunch aosp_cf_x86_64_phone-trunk_staging-userdebug
$ m -j
```

## Kernel + Module Builds

```
$ mkdir kernel && cd kernel
$ repo init -u https://android.googlesource.com/kernel/manifest -b \
    common-android-mainline # or common-android14-6.1
$ repo sync -j
$ tools/bazel run //common:kernel_x86_64_dist \
$ tools/bazel run //common-modules/virtual-device:virtual_device_x86_64_dist
```

## Launch/Interact w/ the device

```
$ cvd start -kernel_path /path/to/bzImage \
    -initramfs_path /path/to/kernel/module/ramdisk
$ adb shell
$ tail -f ~/cuttlefish_runtime/kernel.log // dmesg
> Go to https://127.0.0.1:8443/
```



# Future

- EFI Boot - next presentation :)
- Automotive Virtio SCMI
- Virtio RPMB
- Virtio GPIO
- Media Acceleration (Video Encode/Decode, Camera)

# References

cloud-android-ext@google.com - Feature requests are welcome!

https://source.android.com/docs/setup/create/cuttlefish - for more information

# Generic (Android) Bootloaders

Linaro Connect, May 2024

Ram Muthiah <rammuthiah@google.com>

# What is a typical Android Bootloader?

- A vendor written piece of software that is a part of the final stage in the boot chain and does

- Some things common
  - Fastboot
  - libavb
  - Assembly of the kernel commandline, bootconfig, dt
  - Load into memory of the kernel, ramdisks, bootconfigs, DTBs
  - Device Lock State Assessment
  - Kernel Jump

- Some things not
  - Boot Splash Screens
  - Measured Boot Reporting to TZ
  - RNG
  - Boot Slot Selection
  - Hypervisor Init
  - ... the boot firmware

# A few problems jump out

Every vendor is reimplementing common logic in their boot firmware

- With some vendor specific differences (TZ interactions, Splash Screens, the device firmware underpinning the loader, etc.)

Every release - Android loading requirements can change.

- Init_boot, bootconfig, vendor_boot, etc.
- When they do - firmware developers across the ecosystem update their bootloaders to accommodate.
- But these changes don’t get backported - leaving older SOCs and Devices behind

A lack of updatability

- Anytime a vulnerability is caught in the common load logic, the change has to be backported to 10s-100s of device firmwares

# Past Solutions

- Upstream UBoot
- Android Things
- EFIDroid

# What to do?

If Google could provide a Generic Android Loader that got updates every release, how might a vendor go about integrating it?

And extending it?

A spec conveniently exists for such an interface - UEFI

- The interface is stable and doesn't change frequently (>10 years and counting for 2.10)
- Many bootloaders support it
- It supports discoverable calls

# A proposal - Generic Bootloader (GBL)

Google provides a Generic Android Bootloader EFI Application every release which

- Accommodates new boot requirements
- Gets regular patches for security updates
- Is backwards compatible

The requirement of the vendor to use this - implement a UEFI loader in their final stage which supports the absolute min

- RNG
- Block read/write
- Slot Discovery
- AVB Key Validation
- [link here to source]
- [Your requirements here]

# Getting Started

Code + Readme are here:

https://android.googlesource.com/platform/bootable/libbootloader/+/refs/heads/main/gbl/

Artifacts are here:

https://ci.android.com/builds/branches/aosp_uefi-gbl-mainline/grid

# What’s next

- Getting your feedback and incorporating it
- Looking into LittleKernel UEFI Support
- With that in mind - comments and suggestions are welcome!
  rammuthiah@google.com

# Thank you

Questions?