Linux Kernel Functional Testing (LKFT) 2.0

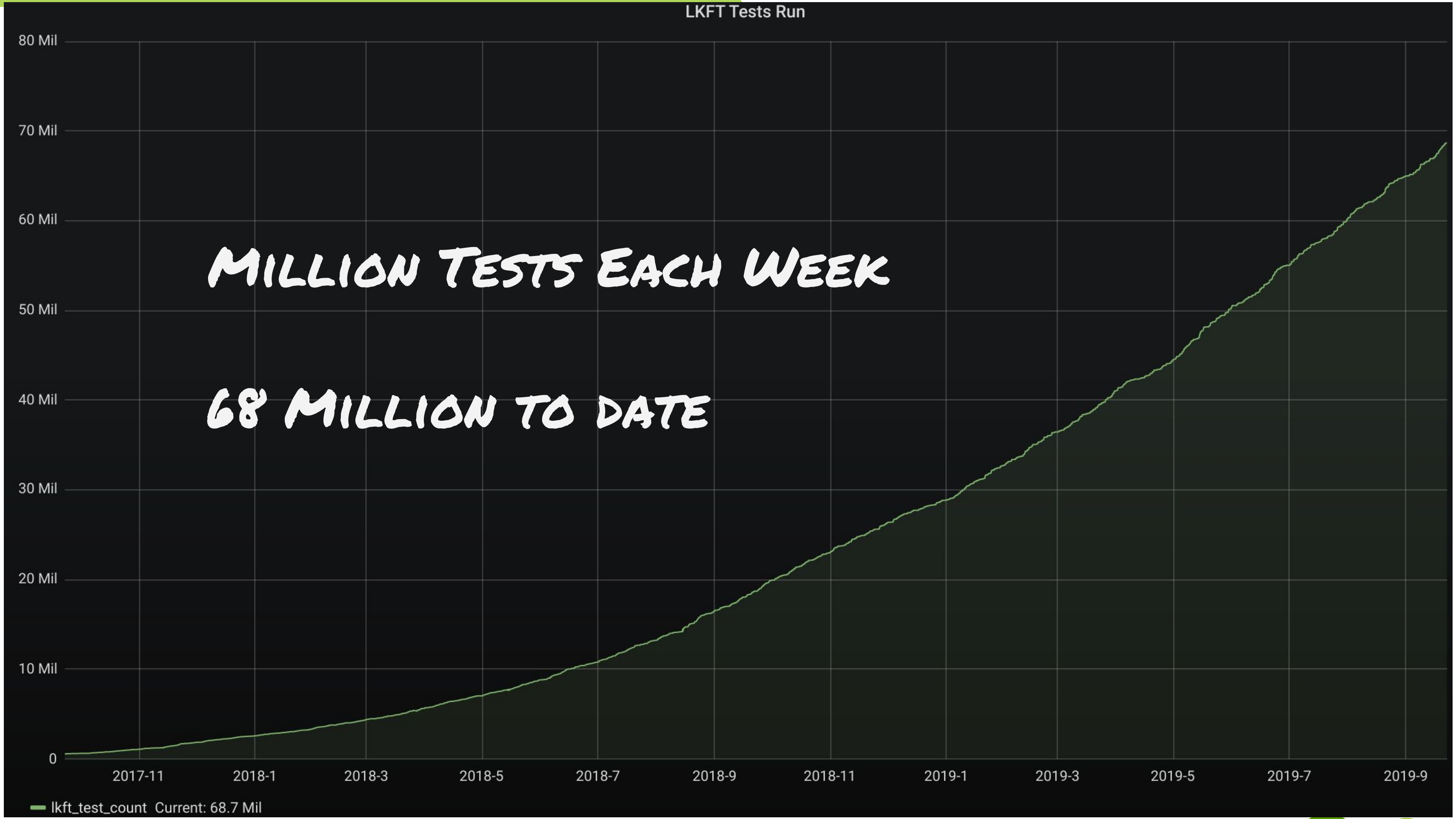
https://lkft.linaro.org
lkft@linaro.org
freenode/#linaro-lkft

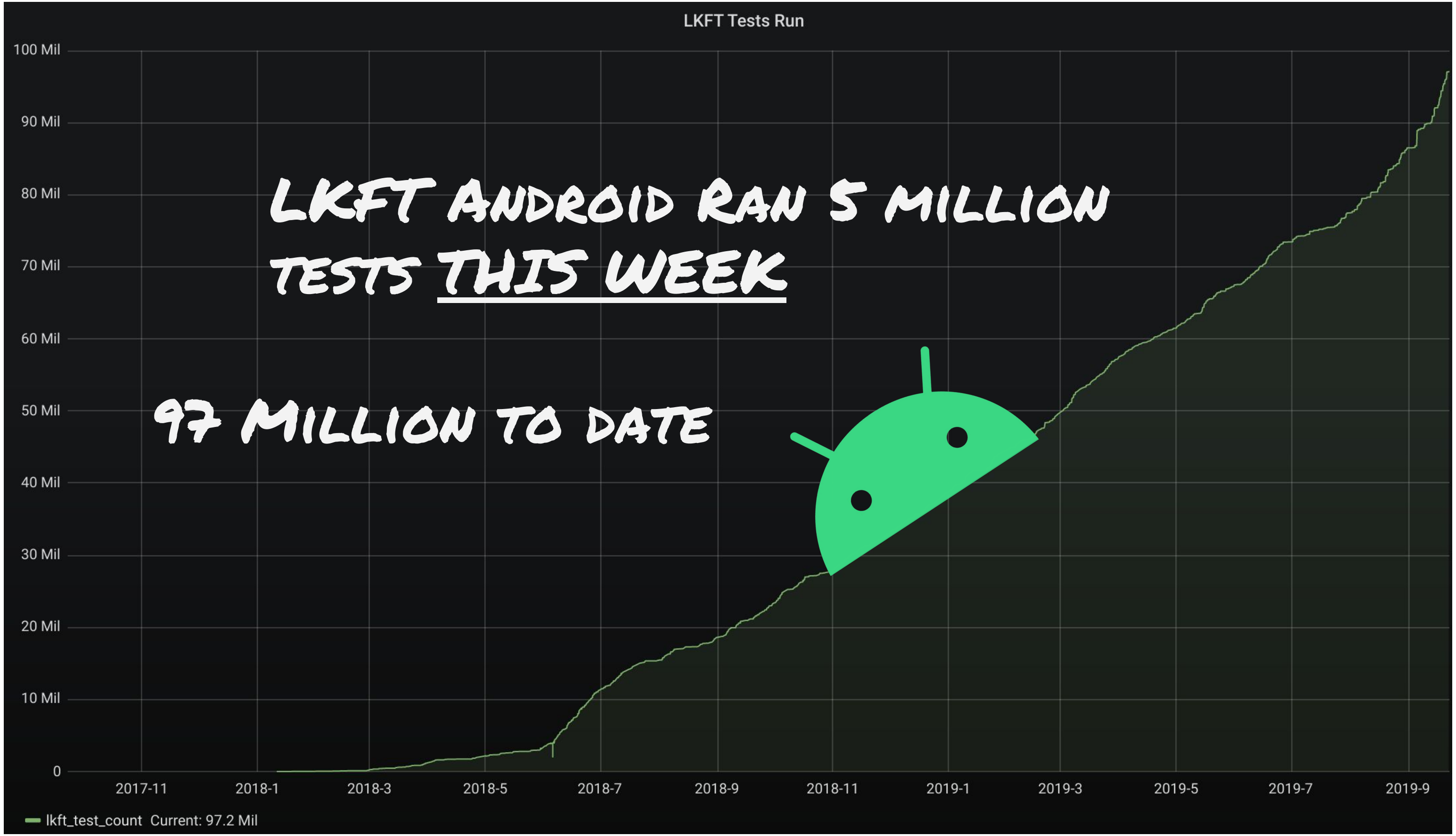
Dan Rue
dan.rue@linaro.org
@mndrue

Linaro connect
San Diego 2019

Intro: LKFT Today

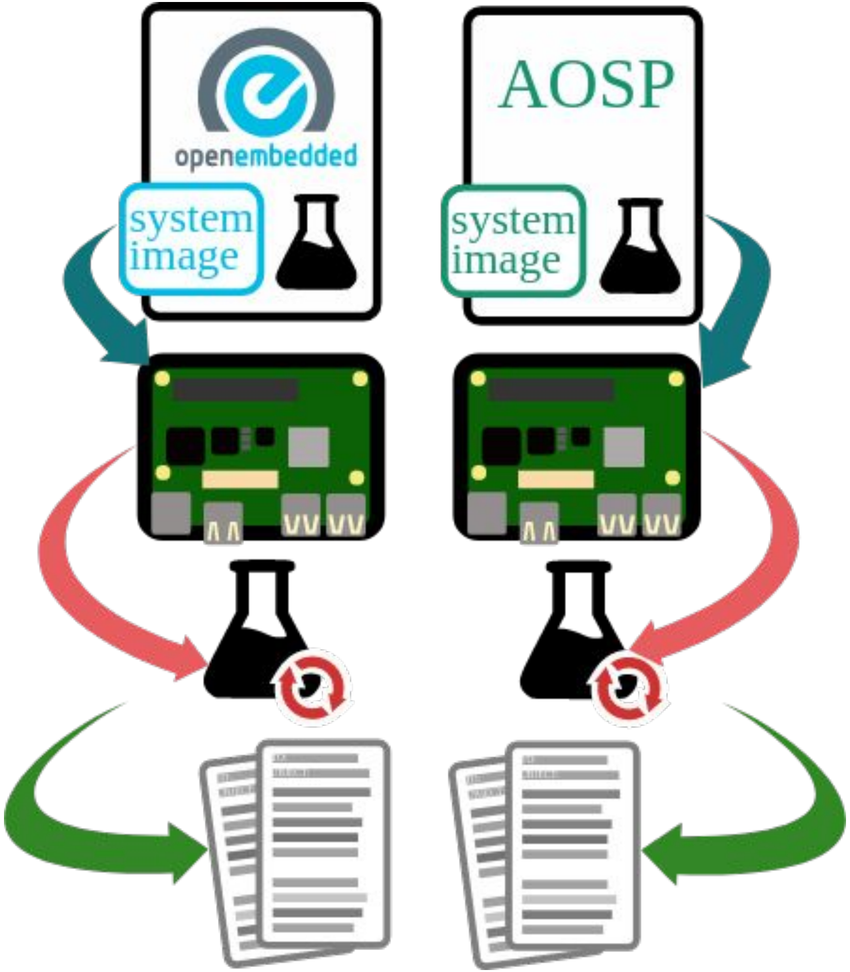
- Architectures: arm32, arm64, i386, x86_64
- Hardware: X15, DragonBoard 410c, Juno, HiKey, x86_64 servers
- QEMU: x86* on x86_64 servers, arm* on SynQuacer arm64 hosts
- Linux Branches:
    - LTS: 4.4, 4.9, 4.14, 4.19
    - Latest stable (5.2, 5.3), mainline, next
- Tests: LTP, libhugetlbfs, perf, v4l2, kvm-unit-tests, s-suite (i/o benchmark), kselftests
- Most tests run in all environments on every push for a total of ~25,000 tests per push.


LKFT Tests Run
Million Tests Each Week
68 Million to date
80 Mil
70 Mil
60 Mil
50 Mil
40 Mil
30 Mil
20 Mil
10 Mil
0
2017-11
2018-1
2018-3
2018-5
2018-7
2018-9
2018-11
2019-1
2019-3
2019-5
2019-7
2019-9
lkft_test_count Current: 68.7 Mil



LKFT

Linaro connect San Diego 2019


openembedded
system image
AOSP
system image

Disclaimer

LKFT 1.0: Build Design

- OpenEmbedded build
- Jenkins based
- Full OS build for every kernel/board combination
- Fixed and shared build capacity
- Build scripts colocated with job config
- Jenkins job file per branch

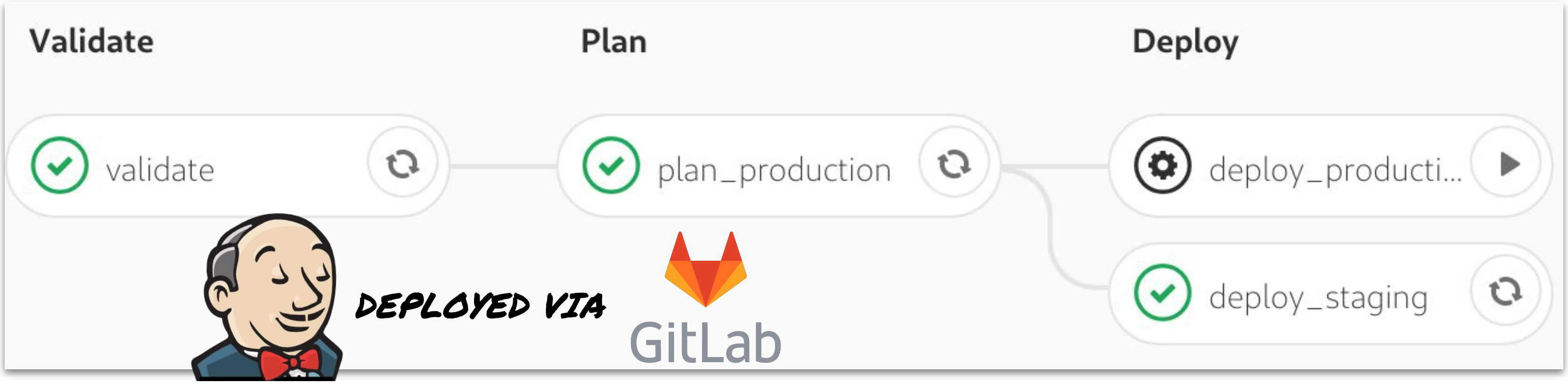
LKFT - Linux Stable RT 4.9 (OpenEmbedded/rocko)	4 mo 0 days - #16	N/A	31 min
LKFT - Linux Stable RT 4.4 (OpenEmbedded/rocko)	19 days - #39	N/A	21 min
LKFT - Linux Stable RC 5.3.y (OpenEmbedded/sumo)	1 day 19 hr - #3	3 days 18 hr - #1	1 hr 15 min
LKFT - Linux Stable RC 5.2.y (OpenEmbedded/sumo)	1 day 19 hr - #70	11 days - #60	37 min

LKFT 1.0: Build Implications

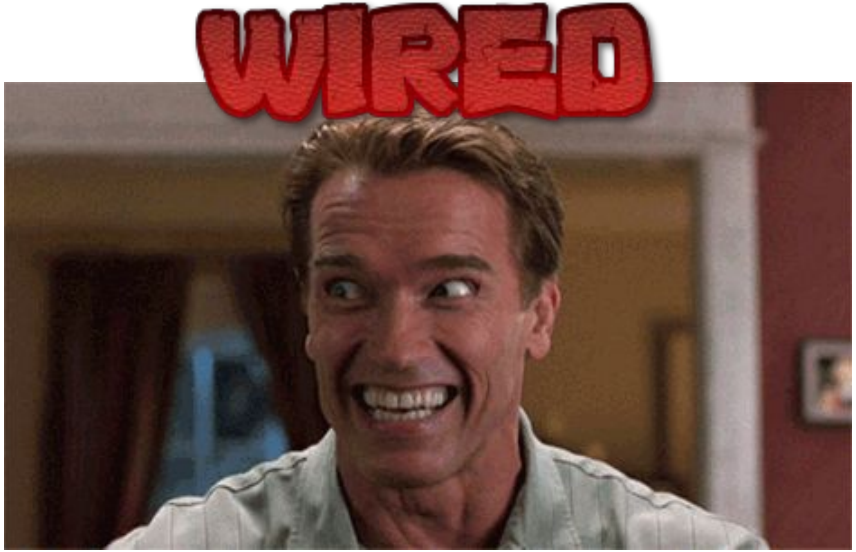
- Builds can be slow
- Builds can be queued
- Ancillary kernels require a full build (e.g. KASAN)... so we don't do them
- Builds are hard to reproduce outside of jenkins environment due to tight coupling
- Changes difficult to test
- Kernel builds use bitbake
    - log is enormous
    - config is derived
    - failures MIGHT be kernel related (but probably aren't)

LKFT 2.0: Build Design

- KernelCI-style builds
  - Root filesystem build independently from kernel
  - Kernel builds are independent and native
  - Docker-based build environments
- Build servers scale dynamically
  - 0 builds, 0 build servers. 20 builds, 20 build servers.
- Artifacts stored and served from cloud storage (s3) directly



LKFT 2.0: Build Implications



- Build times become consistent, and fast
- Ancillary kernels possible and trivial
- Builds are easily reproducible outside of jenkins environment
- Staging environment provides ability to test changes to system
- Kernel build is simple; users will not have to deal with unfamiliar tools
- Artifact hosting is “serverless”
- Compatibility with kernelCI!

LKFT 1.0: Boot Design

- Boards boot using either u-boot or fastboot
- Some boards use system images with kernel baked in
  - X15, qemu_x86/i386 (!), hikey, db410c
- Juno-r2 flashes firmware every run to guarantee correctness
- LAVA job templates colocated with jenkins config

```
- deploy:
    namespace: target
    timeout:
      minutes: 15
    to: tmpfs
    images:
      rootfs:
        image_arg: -drive format=raw,file={rootfs},if=virtio -m 4096 -smp 4 -nographic
        url: http://snapshots.linaro.org/.../rpb-console-image-lkft-intel-corei7-64-20190915215729-2086.hddimg.xz
        compression: xz
    os: oe
```

QEMU x86_64 LAVA Job

LKFT

Linaro connect San Diego 2019

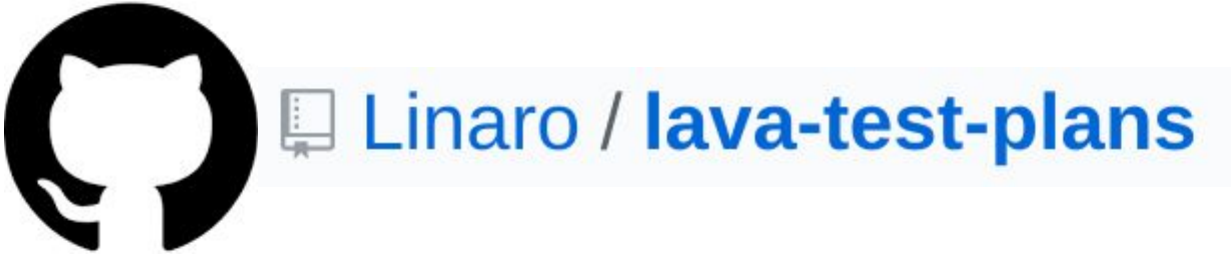
LKFT 1.0: Boot Implications

- Bisection difficult due to per-board and rootfs requirements
- “fastboot flash” slow, and causes contention on dispatcher
- Juno spends 10 minutes re-flashing firmware every run
- LAVA job generation is not portable or reusable (it’s baked in jenkins)

SO TIRED

LKFT 2.0: Boot Design

- LAVA jobs all take a rootfs parameter and a kernel parameter
  - If a baked rootfs is required, it is done in the dispatcher
- Fastboot flash is avoided where possible
- Use NFS based rootfs where possible
- LAVA job generation abstracted to its own tool



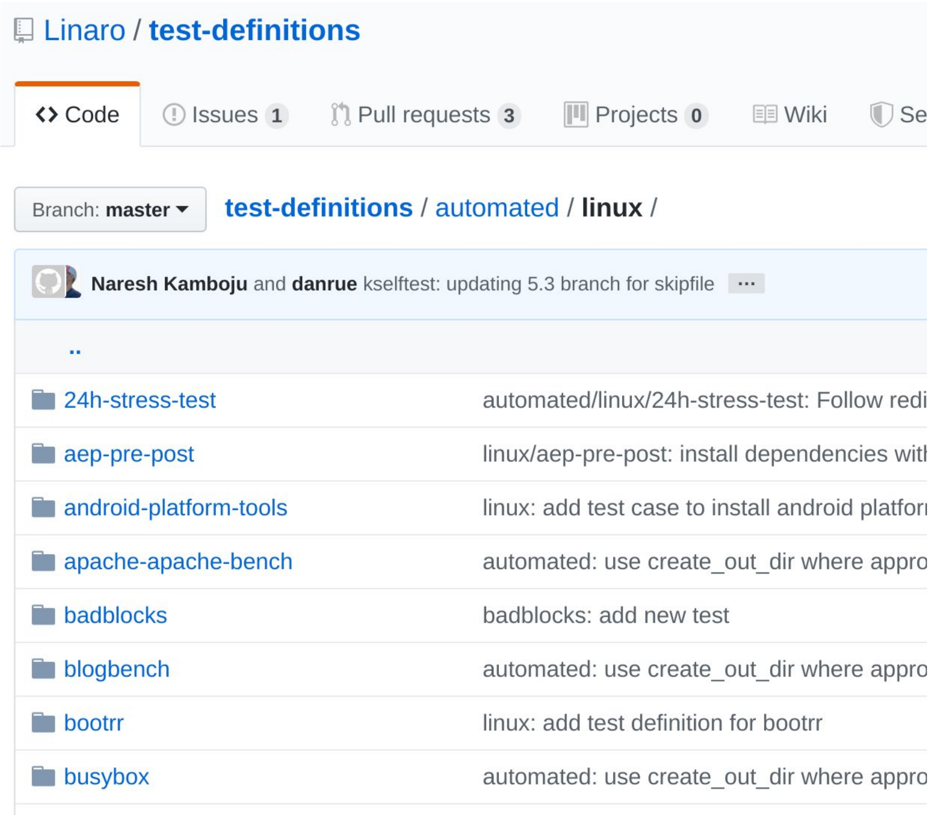
LKFT 2.0: Boot Implications

- Better fastboot provisioning options
    - Network boot when possible
    - inline image building
    - fastboot-nfs
- LAVA job generation is sharable and portable
- Bisection becomes “easy”
- KernelCI compatibility!

LKFT 1.0: Test Design

- Tests generally live in [Linaro/test-definitions](https://github.com/Linaro/test-definitions) on GitHub
- Test binaries usually built into root filesystems
  - Handy for kselftest....
- Single root filesystem for all tests

👏 *test-definitions is general purpose* 👏
Not coupled to LKFT
Not coupled to LAVA



LKFT Linaro connect San Diego 2019

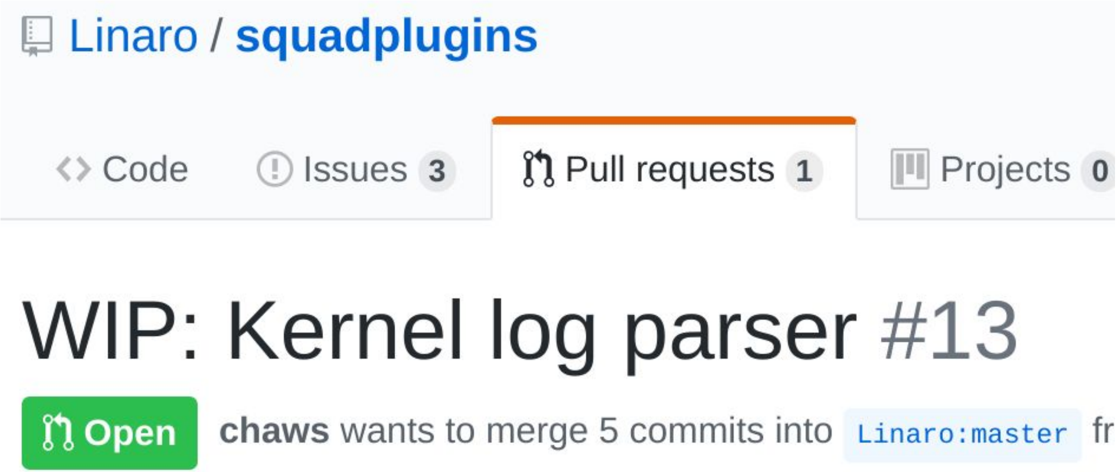
LKFT 1.0: Test Implications

- Space constraints in rootfs (because we only get 1!)

ACTUALLY PRETTY GOOD

LKFT 2.0: Test Design

- Kselftest built along with kernel and overlayed into rootfs via LAVA at runtime
- Possible to have different rootfs for different tests, just as with kernels
- Improved parsing for kernel warnings and errors
- Improved TAP support



LKFT 2.0: Test Implications

- TAP parsing directly in LAVA
- Kernel log parsing in SQUAD

Drink From Data Hose

LKFT 1.0: Report Design

- Template based reports come directly from SQUAD
- Bugs tracked at bugs.linaro.org under product “Kernel Functional Testing”
- Known issues managed in SQUAD to control for failing and flaky tests (see qa-reports-known-issues repo)
- Some reports (stable) generated using SQUAD API and python
- Most upstream reports are manually curated
- Most bugs are manually reported

Jinja is your friend?

```
{%- set ns.test_to_filter_on = 'kselftest' %}
{%- for board,suites in build.test_suites_by_environment.items() %}
 {%- for suite, results in suites -%}
    {%- if ns.test_to_filter_on in suite.slug  %}
     {%- if board not in ns.board_list %}
       {%- set ns.board_list = ns.board_list + [board] %}
     {%- endif %}
   {%- if suite not in ns.suite_list %}
     {%- set ns.suite_list = ns.suite_list + [suite] %}
    {%- endif %}
     {%- set ns.suite_results_pass = ns.suite_results_pass + results['pass'] %}
     {%- set ns.suite_results_xfail = ns.suite_results_xfail + results['xfail'] %}
     {%- set ns.suite_results_fail = ns.suite_results_fail + results['fail'] %}
     {%- set ns.suite_results_skip = ns.suite_results_skip + results['skip'] %}
   {%- endif %}
 {%- endfor -%}
{%- endfor -%}
```

LKFT

Linaro connect San Diego 2019

LKFT 1.0: Report Implications

- Generic reporting - one template/recipient set per branch
- Naive reports - false failures
- Limited ability to provide customization
- Valuable data gathered but stuck in giant database
- Signal:noise ratio not great

LKFT 2.0: Report Design

- Build (or hopefully find!) reporting and analytics layer
    - Perform cross branch and cross time analysis
- Generate fine grained, custom reports
- Support arbitrary frequency
- Integrate with multiple data sources
- Results aggregation? (see kcidb project)
- Automatically identify flaky results. Confidence scoring?

Kernel Validation / KV-238

LKFT Reporting 2.0: Better Scale, Quality, and Efficiency

Linaro / **lkft-tools**

LKFT 2.0: Report Implications

- Achieve high signal:noise ratio
- Support individual developers, and their personal preferences
- Slice data as needed. E.g.
    - Subsystem-specific reports
    - Test-specific reports
    - Board-specific reports

Top 5 Takeaways

💘 kernelCI and LKFT help each other 💘

Kernel builds are as developers expect

Cloud-scale kernel build capacity

Faster and easier to identify root causes of regressions

Reports become more useful to users

tl;dr

We get to say "YES" more.

Thank You!

https://lkft.linaro.org
lkft@linaro.org
freenode/#linaro-lkft

Dan Rue
dan.rue@linaro.org
@mndrue

Linaro connect
San Diego 2019