Linux Kernel Functional Testing (LKFT) 2.0
https://lkft.linaro.org
lkft@linaro.org
freenode/#linaro-lkft

Dan Rue
dan.rue@linaro.org
@mndrue
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.
Million Tests Each Week

68 Million to Date
LKFT Android RAN 5 MILLION TESTS THIS WEEK

97 MILLION TO DATE
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

<table>
<thead>
<tr>
<th>Build</th>
<th>Time</th>
<th>Notes</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>LKFT - Linux Stable RT 4.9 (OpenEmbedded/rocko)</td>
<td>4 mo 0 days - #16</td>
<td>N/A</td>
<td>31 min</td>
</tr>
<tr>
<td>LKFT - Linux Stable RT 4.4 (OpenEmbedded/rocko)</td>
<td>19 days - #39</td>
<td>N/A</td>
<td>21 min</td>
</tr>
<tr>
<td>LKFT - Linux Stable RC 5.3.y (OpenEmbedded/sumo)</td>
<td>1 day 19 hr - #3</td>
<td>3 days 18 hr - #1</td>
<td>1 hr 15 min</td>
</tr>
<tr>
<td>LKFT - Linux Stable RC 5.2.y (OpenEmbedded/sumo)</td>
<td>1 day 19 hr - #70</td>
<td>11 days - #60</td>
<td>37 min</td>
</tr>
</tbody>
</table>
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

DEPLOYED VIA

GitLab
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

```yaml
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/.../rpbc-console-image-lkft-intel-corei7-64-20190915215729-2086.hddimg.xz
      compression: xz
  os: oe
```

QEMU x86_64 LAVA Job
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)
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) 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 1.0: Test Implications

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

```latex
\%
\set ns.test_to_filter_on = 'kseltest'
\%
\for board, suites in build.test_suites_by_environment.items()
\%
\set ns.board_list = ns.board_list + [board]
\%
\if suite not in ns.board_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']
\%
\endfor
\%
\end
```
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