Testing your AGL, yocto ptest, lava and more

ALS 2018

Jan-Simon Möller
Release Manager, AGL , The Linux Foundation

jsmoeller@linuxfoundation.org,
DL9PF @IRC and elsewhere

AUTOMOTIVE GRADE LINUX

Image: public domain

Dipl.-Ing. Jan-Simon Möller

jsmoeller@linuxfoundation.org

'DL9PF' on #freenode

AGL Release Manager, EG CIAT Lead

AUTOMOTIVE GRADE LINUX

Introduction

Platform and Applications in AGL

- Platform
  - Base system incl. libraries
  - Built with the Yocto Project
  - Application framework
  - Other middleware

  → Part of filesystem image

- Applications & Services
  - Services provide APIs
  - Applications consume APIs
  - Built with SDK
  - Packaged as .wgt

  → Installed at runtime.

AUTOMOTIVE GRADE LINUX

What to do where ?

- You work on the Platform if you deal with a:
  - system library
  - kernel driver
  - BSP
  - framework (itself)

→ low level

- You work on the Applications/Services if you deal with a:
- Service (agl-service-*)
- Application

→ high level

"Platform"

- The outcome here is usually a **filesystem image** but it can also be a **package feed**
- We have two options to inject tests in the process
  - 'Early' as compile-time tests
    - Actually a great option as we get feedback very early – at compile-time
    - But this usually does not work well as we're cross-compiling and cannot execute the generated binaries
  - 'Late' once the image is created and booted
    - This works well but requires the target to be deployed and booted
    - For CI this needs to be automated

"Applications & Services"

- The outcome of the compilation is a ***.wgt file**
- Code is compiled for the target arch
- wgt files need to be **installed at runtime** (dynamic IDs / smack labels for security)
- Thus tests need to be **executed at runtime**

Scope

- Let's explore
  - How to add tests to AGL 'Platform'
  - How to add tests to AGL 'Apps / Services'
  - How to run the tests on the target
  - What automation framework can be used
  - and how reporting is done

How to add tests to the AGL 'platform'

AUTOMOTIVE GRADE LINUX

Platform (1)

- The Platform is built using the YP
- As discussed – compile-time tests would allow as to fail early , but we cannot execute the code if cross-compiled
- But what can we do:
  - system libraries and programs usually come with a testsuite (aka 'make test')
  - you have your own testsuite ?
  - let's use it !

Platform (2)

- The YP has a feature for this called **ptest**
- In principle a **ptest** is the 'make test' packaged
- It can then be deployed on the target and executed using *ptest-runner*

Platform (3)

from zlib_1.2.11.bb:

```
SRC_URI += "file://run-ptest"
inherit ptest
do_compile_ptest() {
    oe_runmake test
}
do_install_ptest() {
    install ${B}/Makefile   ${D}${PTEST_PATH}
    install ${B}/example    ${D}${PTEST_PATH}
    install ${B}/minigzip   ${D}${PTEST_PATH}
    install ${B}/examplesh  ${D}${PTEST_PATH}
    install ${B}/minigzipsh ${D}${PTEST_PATH}

    # Remove buildhost references...
    sed -i -e "s,--sysroot=${STAGING_DIR_TARGET},,g" \
        -e 's|${DEBUG_PREFIX_MAP}||g' \
     ${D}${PTEST_PATH}/Makefile
}
RDEPENDS_${PN}-ptest += "make"
```

wrapper script for target

compilation procedure for testsuite

install test binaries

adapt scripts/path to target execution if necessary

declare (undetectable) runtime dependencies for tests (e.g. make)

AUTOMOTIVE GRADE LINUX

Platform (4)

- How is it added to the filesystem ?
  - To add package testing to your build,
    set the **DISTRO_FEATURES** and **EXTRA_IMAGE_FEATURES**

```
DISTRO_FEATURES_append = " ptest"
EXTRA_IMAGE_FEATURES += "ptest-pkgs"
```

- Shorthand is the **agl-ptest** feature for aglsetup.sh
- All ptest files are installed in
  /usr/lib/<package>/ptest

Platform (5)

- How is it executed ?
- The "ptest-runner" package installs a "ptest-runner" which loops through all installed ptest test suites and runs them in sequence.

How to add tests to AGL 'Apps / Services'

AUTOMOTIVE GRADE LINUX

Applications and Services (1)

- For the applications and services, we actually face multiple areas
  - we need to test the highlevel API calls of the services
  - we need to test the applications
  - we want reports on the code coverage

Applications and Services (2)

- For testing the highlevel calls, there is work in progress to use lua scrips for this task:
- https://github.com/iotbzh/afb-test
  - https://github.com/iotbzh/afb-test/blob/master/README.md
  - https://github.com/iotbzh/afb-test/tree/master/conf.d/project/lua.d

- Final goal:
  add it as part of the application-templates

Applications and Services (3)

- gcov based code-coverage reporting
  - requires a separate build / binary
  - executed on the targed, produces *.gcov files for each source file
- Work done to integrate this also into the makefiles of the app templates as well.

Applications and Services (4)

- Common to all:
  - they need to be executed on the target
  - partially with performance penalty (gcov)
  - for automation, this means we add a wrapper script to each service or application to exec the procedure
  - This is being called similar or equal to the ptest-runner
  - Executed in the CIAT infra

How to run the tests on the target

How to run it on the target (1)

- Manual:
  - Platform:
    - ptest: either by ptest-runner or
      call run-ptest script directly
    - All ptest files are installed in /usr/lib/<package>/ptest
  - Applications/Services
    - wrapper script required as entry point for CI (alike ptest)
    - tbd if this is part of app-templates

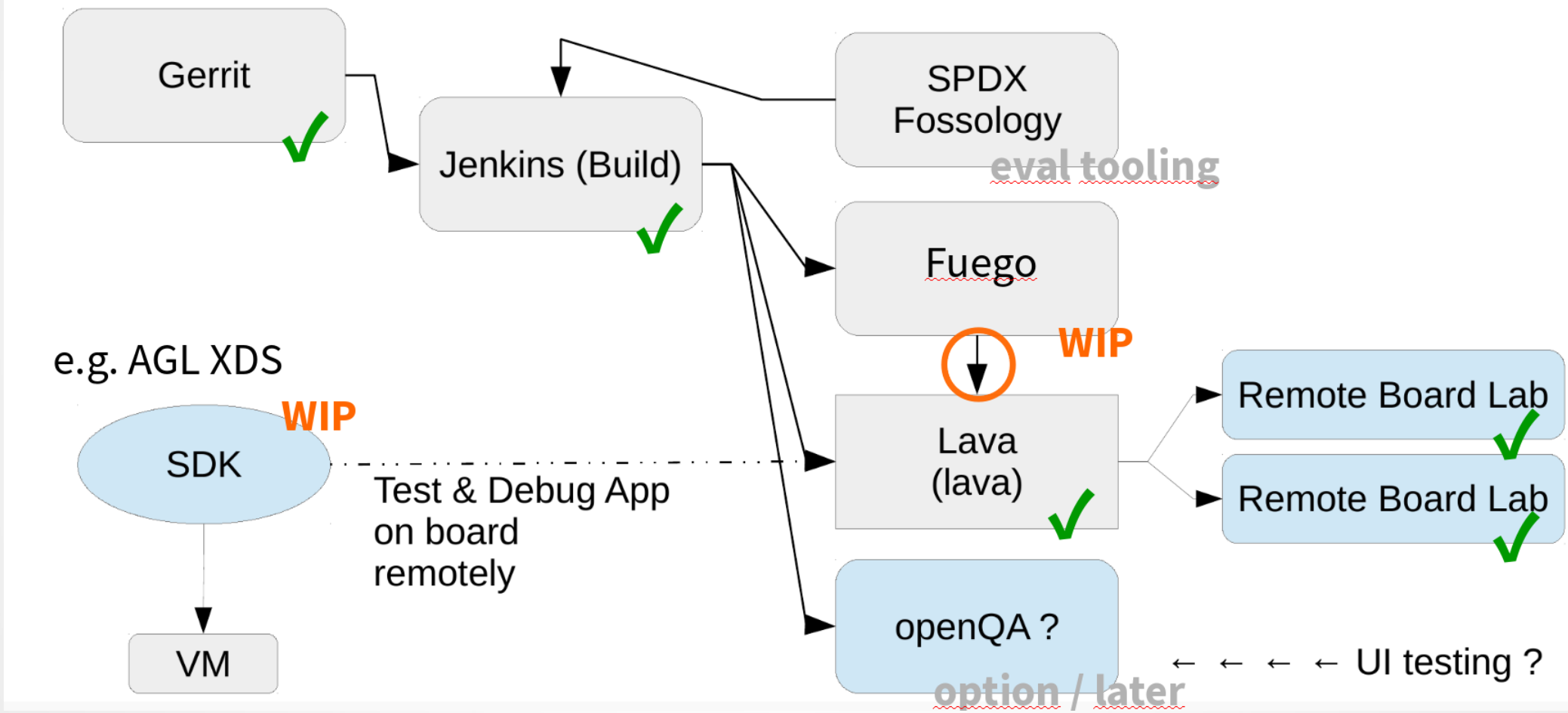
How to run it on the target (2)

- Common issues:
  - needs to run on target
  - we need a common reporting
    - agreement is to use the KernelCI/Fuego json format
    - alternative: tap
- Join the conversation and the upcoming calls

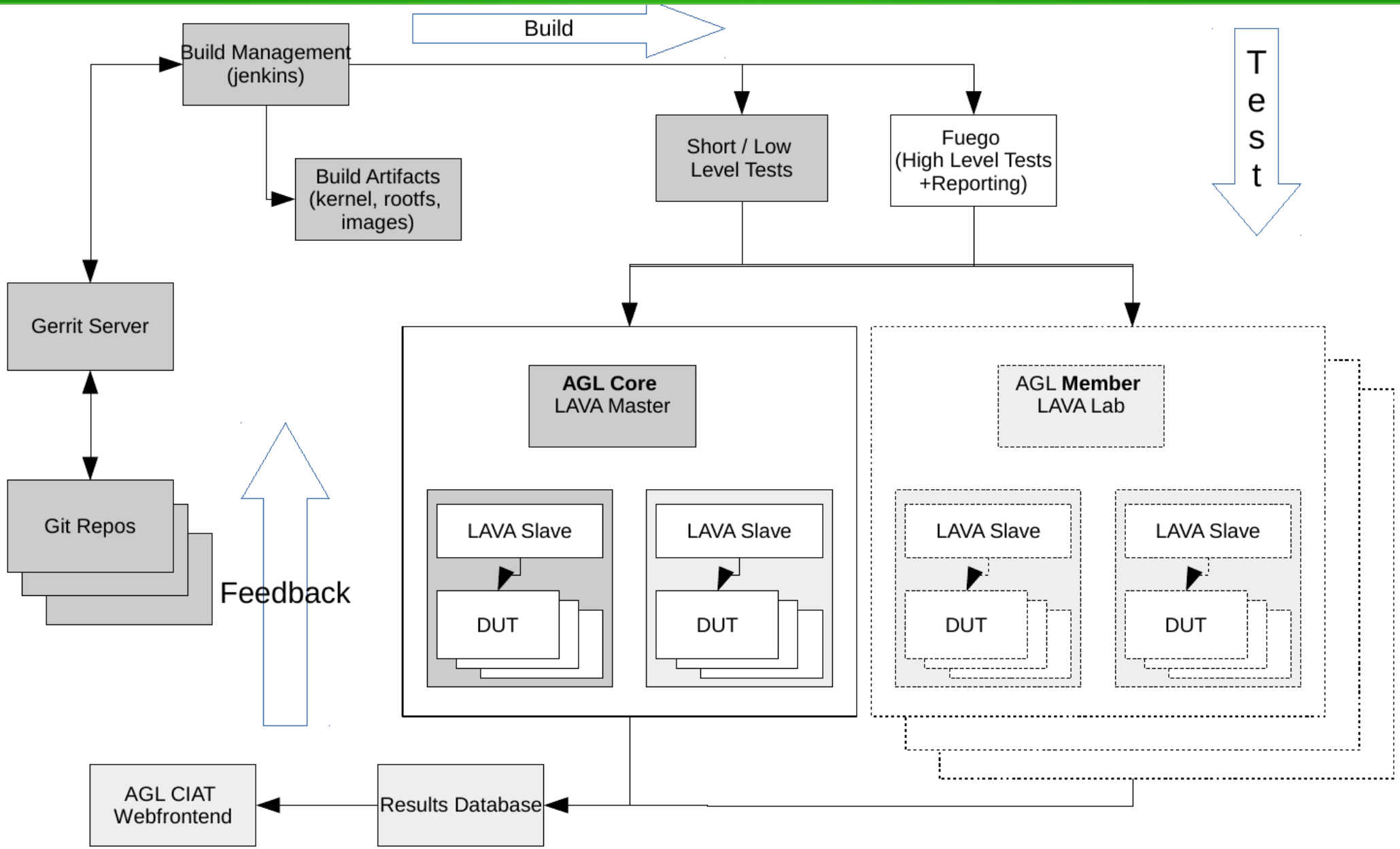
What automation framework(s) can be used

A look back … :

A Vision/Plan (mid/long)

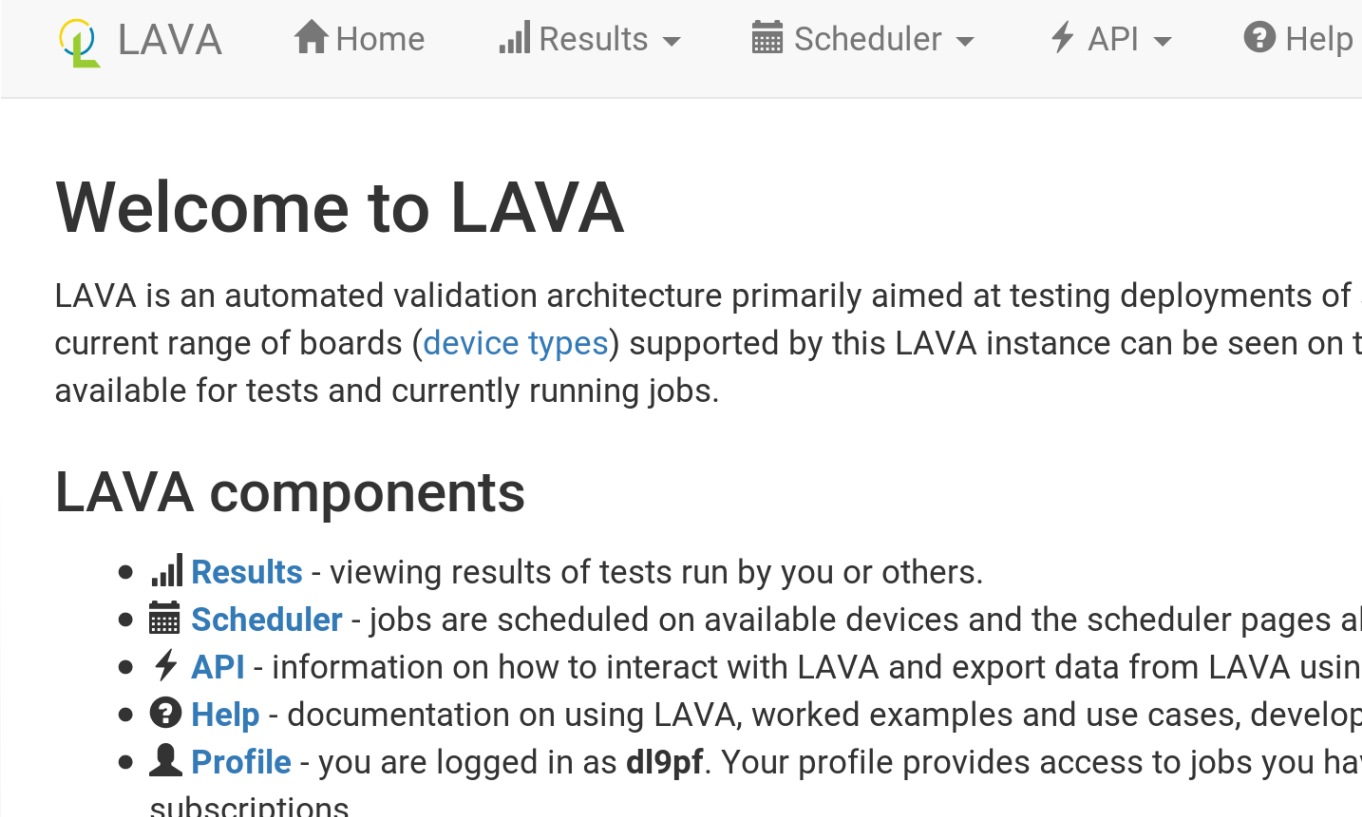


The AGL CI infra overview

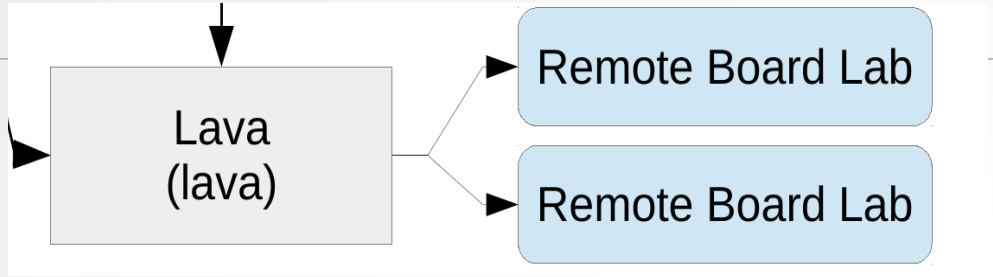


LAVA

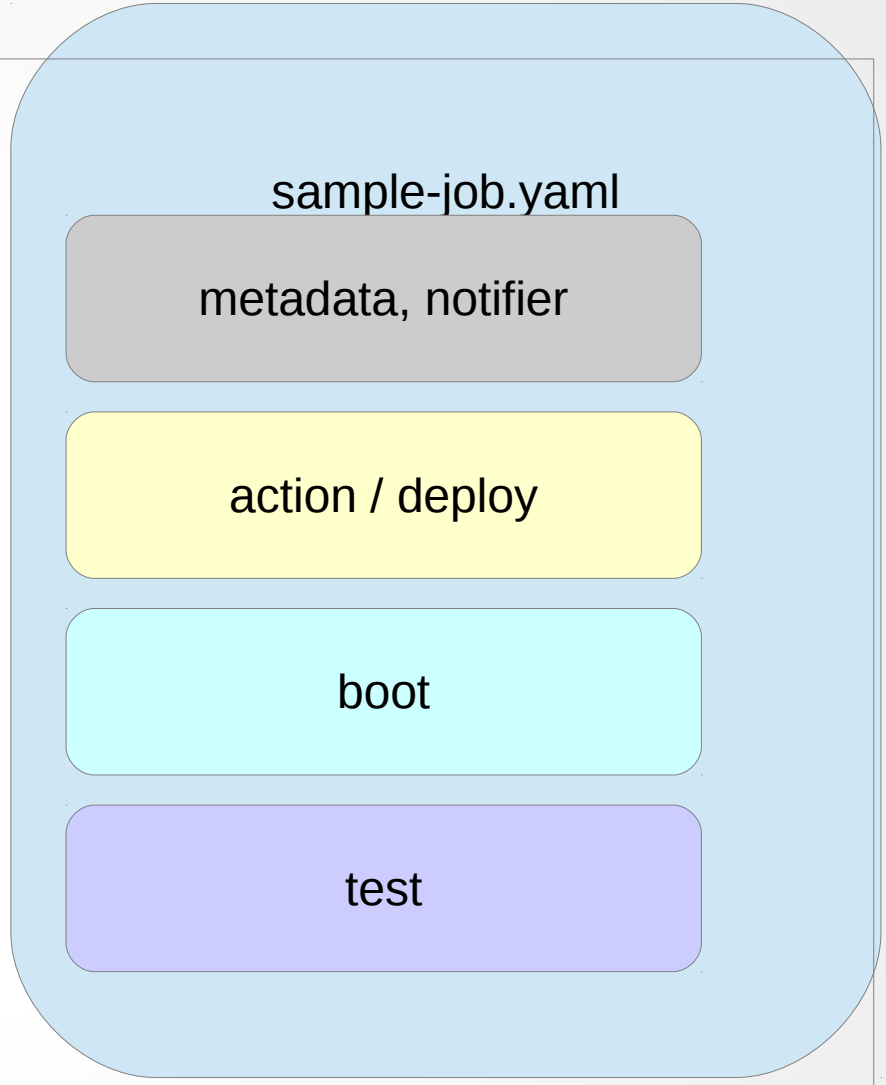
- AGL uses LAVA and hosts an instance on https://lava.automotivelinux.org
- Current remote labs:
  - lab-AGL-core
  - lab-baylibre
  - lab-iotbzh



LAVA



- metadata for the job
- action/deploy section
  - files to be used
- boot section
- test section



Test section

- One or multiple
  - inline
  - from git repo
  - uses yaml files
  - lava-test-* are markers
    - for visualizing in LAVA
    - for visualizing in kernelCI
    - for cross-referencing

```yaml
- test:
    timeout:
      minutes: 2
    definitions:
    - repository:
        metadata:
          format: Lava-Test Test Definition 1.0
          name: inline-test
          description: "Inline test to validate test framewrok health"
          os:
          - debian
          scope:
          - functional
        run:
          steps:
          - lava-test-set start set-pass
          - lava-test-case always-pass --shell true
          - lava-test-set stop set-pass
          - lava-test-set start set-fail
          - lava-test-case always-fail --shell false
          - lava-test-set stop set-fail
      from: inline
      name: health-test
      path: inline/health-test.yaml

- test:
    definitions:
    - repository: https://git.automotivelinux.org/src/qa-testdefinitions
      from: git
      path: test-suites/short-smoke/busybox.yaml
      name: busybox
    - repository: https://git.automotivelinux.org/src/qa-testdefinitions
      from: git
      path: test-suites/short-smoke/smoke-tests-basic.yaml
      name: smoke-tests-basic
    - repository: https://git.automotivelinux.org/src/qa-testdefinitions
```

Test section details (inline/git)

```
- test:
    [..]
    definitions:
        - repository:
            metadata:
                format: Lava-Test Test Definition 1.0
                name: smoke-tests-basic
                description: "Basic test command for AGL images"
            run:
                steps:
                    - agl-basic-test-shell-command
          from: inline
          name: agl-dut-inline-basic
          path: inline/agl-dut-inline-fake-filename.yaml
        - repository: git://git.automotivelinux.org/src/qa-testdefinitions.git
          from: git
          path: test-suites/short-smoke/smoke-tests-basic.yaml
          name: smoke-tests-basic
        - repository: https://git.linaro.org/lava-team/lava-functional-tests.git
          from: git
          path: test-suites/short-smoke/service-check.yaml
          name: service-check
```

AUTOMOTIVE GRADE LINUX

Example: add a 'systemd service up' check

- https://git.automotivelinux.org/src/qa-testdefinitions/tree/test-suites/short-smoke/service-check.yaml

```
[...]
run:
    steps:
        - "cd common/scripts"
        - "./service-check-gfx.sh"
```

Example: add a 'systemd service up' check

- https://git.automotivelinux.org/src/qa-testdefinitions/tree/common/scripts/service-check-gfx.sh

```
#!/bin/bash

export LANG=C
export TERM=dumb

REQUIREDSOCKETS="cynara.socket dbus.socket security-manager.socket"
REQUIREDSERVICES="afm-system-daemon.service connman.service ofono.service weston.service homescreen.service bluetooth.service"

ALL="${REQUIREDSOCKETS} ${REQUIREDSERVICES}"
RESULT="unknown"

# add delay for services to fully start
sleep 5

for i in ${ALL} ; do
    echo -e "\n\n########## Test for service ${i} being active ##########\n\n"

    systemctl is-active ${i} >/dev/null 2>&1
    if [ $? -eq 0 ] ; then
        RESULT="pass"
    else
        RESULT="fail"
    fi

    lava-test-case ${i} --result ${RESULT}
    systemctl status ${i} || true
    echo -e "\n\n"

    echo -e "\n\n########## Result for service ${i} : $RESULT ##########\n\n"
done
```

AUTOMOTIVE GRADE LINUX

Now its your turn:

- We need you to add your service checks !
  - in above script
- We need you to add your testsuites !
  - in qa-testdefinitions

- More details in my talk from AMM 2017 !

and how reporting is done

KernelCI

- We use KernelCI to present the results
- https://kernelci.automotivelinux.org
- e.g.: https://kernelci.automotivelinux.org/test/board/r8a7796-m3ulcb/job/AGL-kernel-tree/kernel/AGL-gerrit-14179-1/

KernelCI (2)

Home | Jobs | Builds | Boots | SoCs | Tests β | Compare β | Info

Details for Tree «AGL-kernel-tree» - AGL-gerrit-14179-1

0 / 0 / 0

Board	r8a7796-m3ulcb
Tree	AGL-kernel-tree — -
Git branch	agl-branch
Git describe	AGL-gerrit-14179-1 — -
Git URL	⊘
Git commit	⊘
Date	2018-06-05

«AGL-core-lab-1»

25 reports per page

Test suite name	Test suite ID	Total test sets	Test Results
busybox	5b161a6d19bd3200370cad8f	1	1 1 0 0
service-check	5b161a6c19bd3200370cad84	1	9 6 3 0
smoke-tests-basic	5b161a6b19bd3200370cad7d	1	5 4 1 0
health-test	5b161a6a19bd3200370cad78	2	2 1 1 0
yocto-ptest	5b161a6919bd3200370cad75	1	1 0 0 1
lava	5b161a6719bd3200370cad51	1	34 34 0 0

Test details for test suite «service-check» (AGL-core-lab-1)

Lab name	AGL-core-lab-1 fl
Board	r8a7796-m3ulcb fl
Tree	AGL-kernel-tree —
Git branch	agl-branch
Git describe	AGL-gerrit-14179-1 —
Defconfig	defconfig+CONFIG_AGL=y
Date	2018-06-05 05:06:52 UTC

Status	⊘
Architecture	arm64
Errors	⊘
Warnings	⊘
Test time	0
Boot & Test log	txt — html

Test Reports

Test set: default

Test Case Name	Measurements	Date	Status
bluetooth.service	∅	2018-06-05	⚠
homescreen.service	∅	2018-06-05	⚠
weston.service	∅	2018-06-05	⚠
ofono.service	∅	2018-06-05	✓
connman.service	∅	2018-06-05	✓
afm-system-daemon.service	∅	2018-06-05	✓
security-manager.socket	∅	2018-06-05	✓

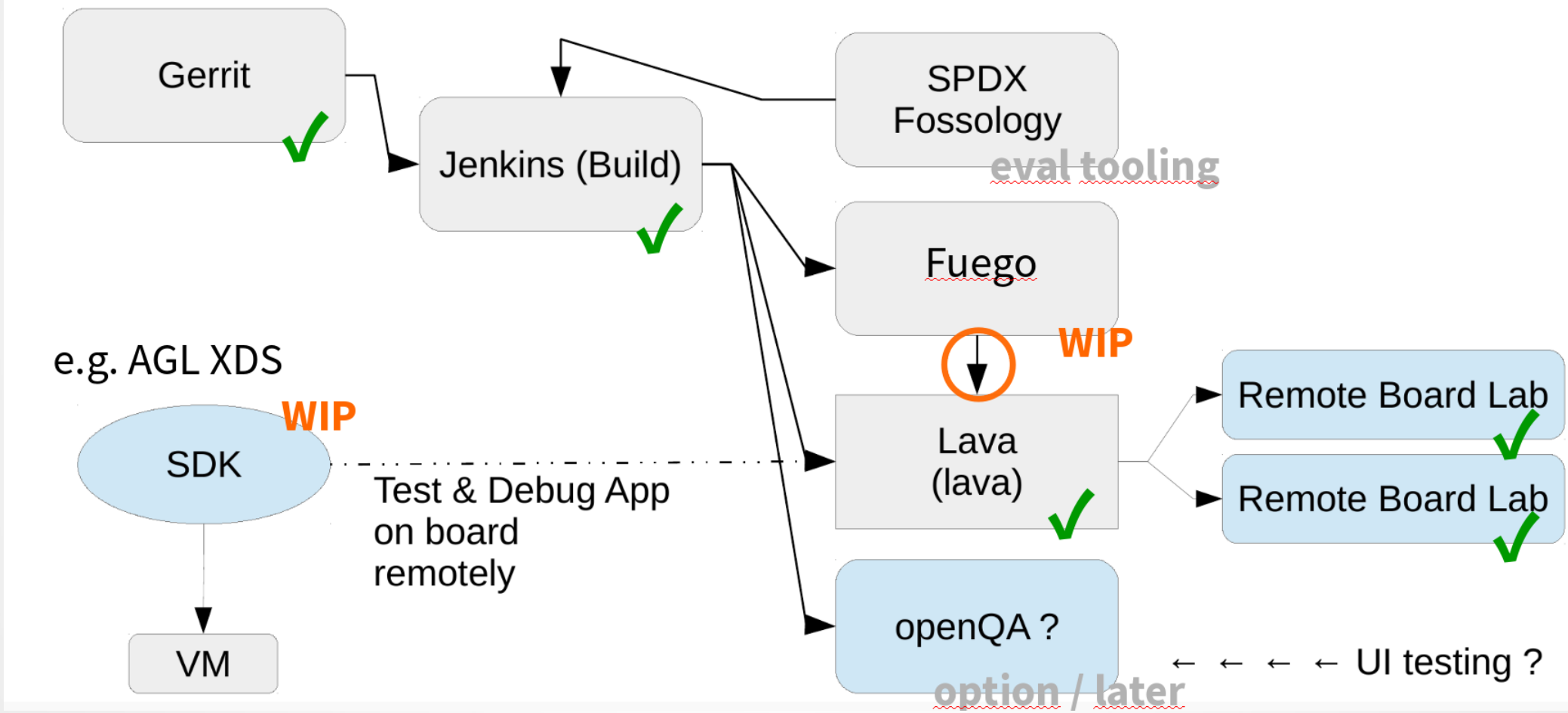
KernelCI (3)

- Next steps:
  - enhance WebUI
  - Cross-references

Whats next ?

Filling the gaps :

A Vision/Plan (mid/long)



QA

Thank you.

Contact:

jsmoeller@linuxfoundation.org

AUTOMOTIVE GRADE LINUX

References

- 2017 AMM Talk on writing new tests: http://bit.ly/2ll5SVy
- ptest: https://wiki.yoctoproject.org/wiki/Ptest
- gcov wip: http://bit.ly/2M4CWMQ
- Writing tests for lava: http://bit.ly/2ywcDgQ