Tutorial: Writing a LAVA Test Definition and Running it in the AGL Infra

AMM Dresden 2018

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

jsmoeller@linuxfoundation.org,
DL9PF @IRC and elsewhere

Dipl.-Ing. Jan-Simon Möller

jsmoeller@linuxfoundation.org

'DL9PF' on #freenode

AGL Release Manager, EG CIAT Lead

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.

What to do where ?

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

→ low level

(platform point of view)

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

→ high level

(platform point of view)

AUTOMOTIVE GRADE LINUX

"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 one ore multiple ***.wgt file(s)**
- 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 the AGL 'platform'
  - How to add tests to AGL 'Apps / Services'
  - How to run the tests on the target
  - Let's start small – inline definitions
  - Test definitions from a git repo

How to add tests to the AGL 'platform'

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'

CAUTION
Work in progress
CLOSING FUSE
4 AMP

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 application logic ~~((and UI))~~
  - 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:
  - afb-test
- gcov based code-coverage reporting available as well

- Final goal:
  make this part of every reference app and run for each changeset

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 through a qa-testdefinition
  - 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
      - needs to be in predefined location /usr/share/agl-test/<pgkname>.sh
    - tbd if this is part of the app templates
    - of course manual runs on the terminal or shell as well

How to run it on the target (2)

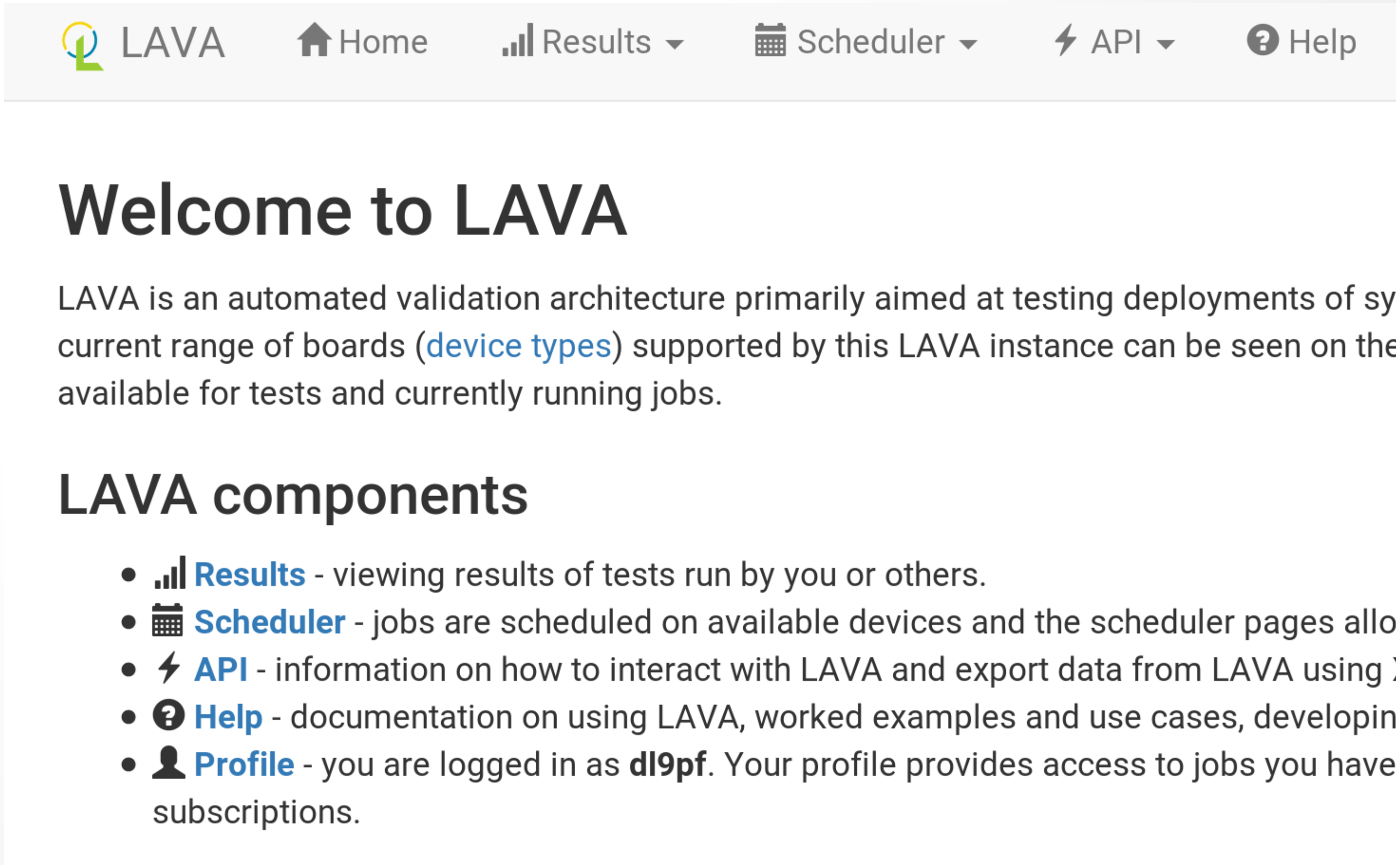
- Common issues:
  - needs to run on target
  - we need a common reporting
    - agreement is to use the KernelCI/Fuego json format
    - alternative: tap

LAVA

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

- Account requests via JIRA only

  (no LFID)



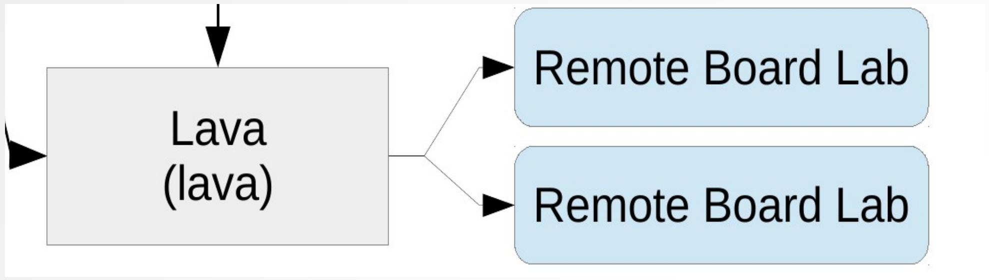
Welcome to LAVA

LAVA is an automated validation architecture primarily aimed at testing deployments of sy
current range of boards (device types) supported by this LAVA instance can be seen on the
available for tests and currently running jobs.

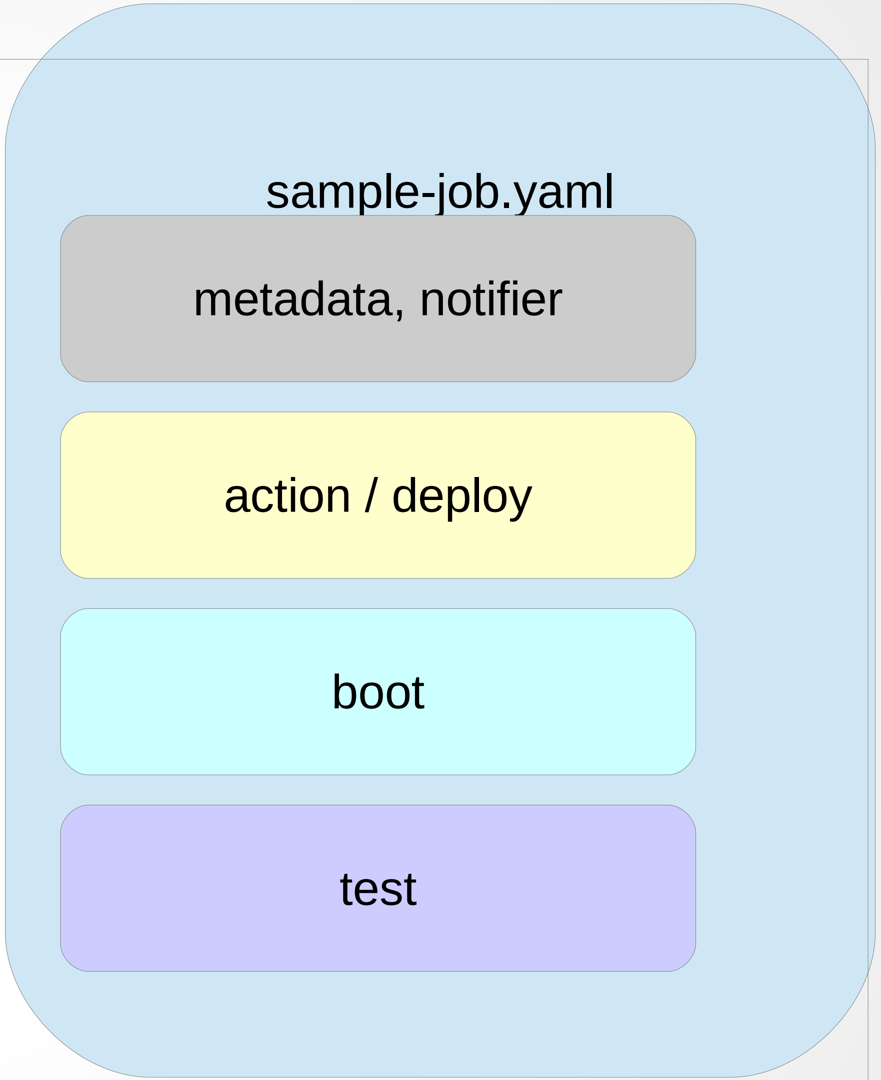
LAVA components

- **Results** - viewing results of tests run by you or others.
- **Scheduler** - jobs are scheduled on available devices and the scheduler pages allo
- **API** - information on how to interact with LAVA and export data from LAVA using
- **Help** - documentation on using LAVA, worked examples and use cases, developin
- **Profile** - you are logged in as **dl9pf**. Your profile provides access to jobs you have subscriptions.

LAVA Job Definition

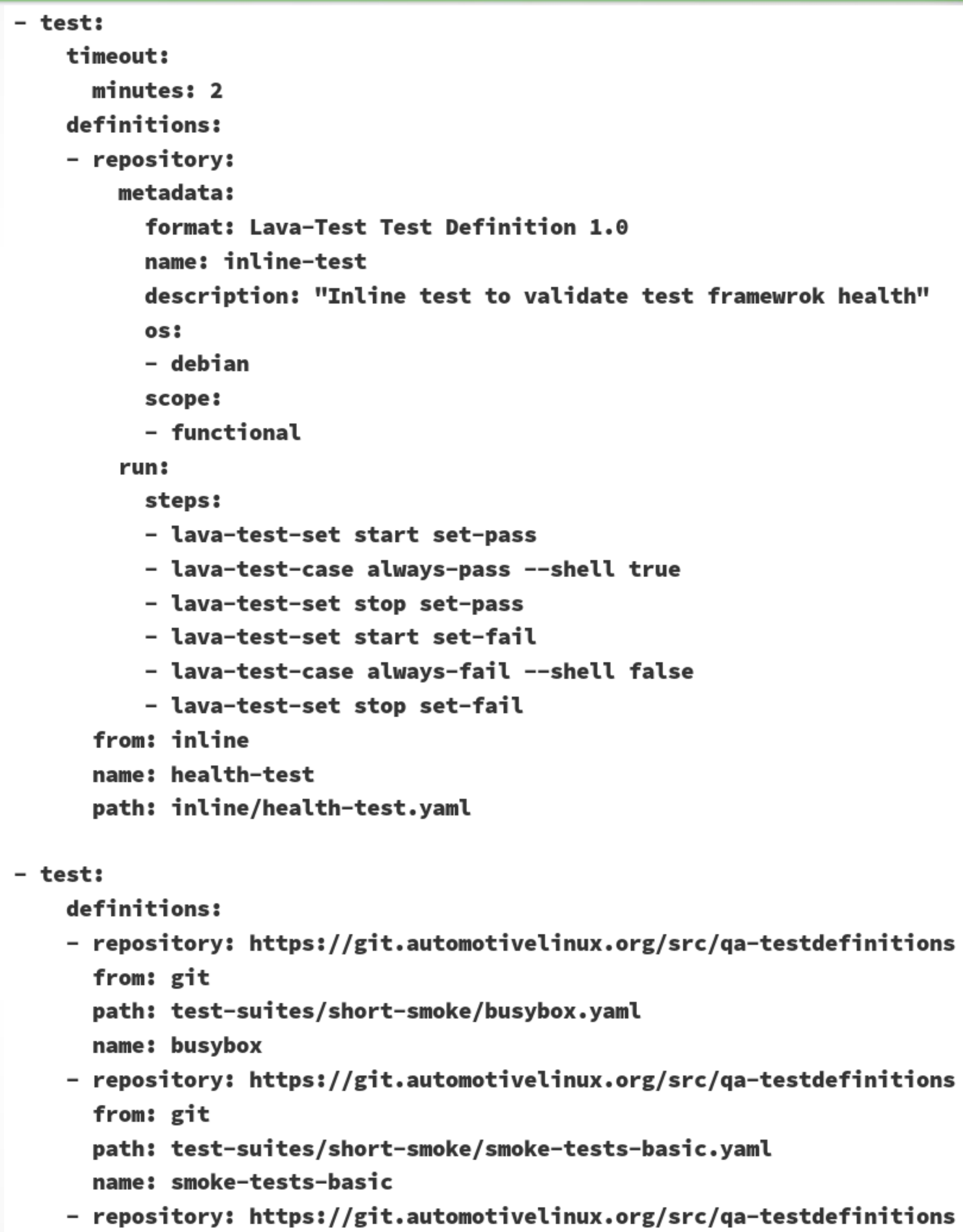


- Is a yaml style file
- contains
  - metadata for the job
  - action/deploy section
    - files to be used
  - boot section
  - test section



Test section

- One or multiple
  - inline
  - from git repo
  - uses yaml files
  - lava-test-* are markers
    - required for processing
    - used also for cross-referencing



```
- 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
```

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

```bash
#!/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
```

Now its your turn:

- We need **you** to add your service checks !
  - in above script
- We need **you** to add your testcases !
  - in qa-testdefinitions
    - e.g. can
    - e.g. audio playback / reception
    - e.g. app lifecycle (install/uninstall/start/stop)

- Next, let's construct two examples and run them on lava.automotivelinux.org

Let's start small – inline definitions

inline testdefinition

- All is part of the lava job definition (this 'inline')
- Quick and easy, but only usefull for development & debuging & ad-hoc

inline testdefinition

```
- test:
    timeout:
      minutes: 4
    definitions:
    - repository:
        metadata:
          format: Lava-Test Test Definition 1.0
          name: sample-inline-test
          description: "Sample inline test definition"
          os:
          - oe
          scope:
          - functional
        run:
          steps:
          - lava-test-case ls --shell ls /
          - lava-test-case os-release --shell cat /etc/os-release
      from: inline
      name: sample-inline-test
      path: inline/sample-inline-test.yaml
```

- Let's try:
  - https://lava.automotivelinux.org/scheduler/job/917/resubmit

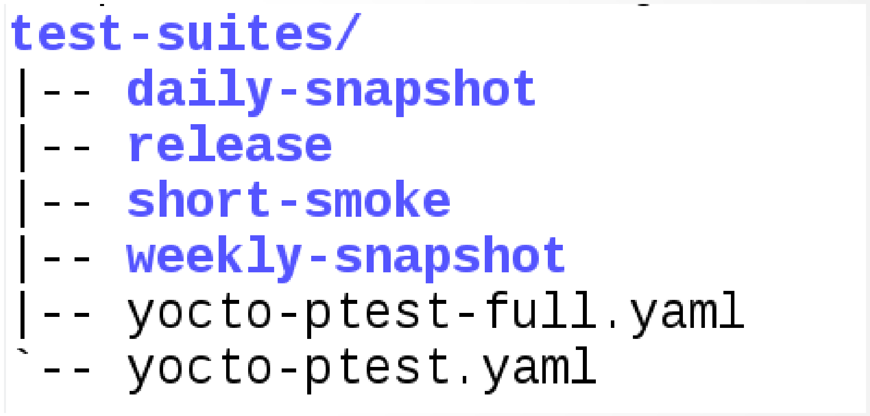
Test definitions from a git repo

Testdefinition from git

- Lava job definition references git repo and test yaml
- Can be shared and repeated across multiple instances of lava
- Persistent across single lava jobs

Tests from a git repo

- AGL hosts such a repo under
  - src/qa-testdefinitions
- Other sources are e.g.
  - https://git.linaro.org/qa/test-definitions.git



```
test-suites/
|-- daily-snapshot
|-- release
|-- short-smoke
|-- weekly-snapshot
|-- yocto-ptest-full.yaml
`-- yocto-ptest.yaml
```

sample-test-from-repo

Let's write a sample yaml

```
metadata:
    format: Lava-Test Test Definition 1.0
    name: sample-test
    description: "Sample"
    maintainer:
        - your.name@isp.net
    os:
        - openembedded
    scope:
        - functional

run:
    steps:
        - lava-test-case sample-pwd --shell pwd
        - lava-test-case sample-uname --shell uname -a
```

sample-test-from-repo

Or the steps-script is part of the git repo as well:

```
metadata:
    format: Lava-Test Test Definition 1.0
    name: sample-test-script
    description: "Sample with script"
    maintainer:
        - your.name@isp.net
    os:
        - openembedded
    scope:
        - functional

run:
    steps:
        - "cd common/scripts"
        - "./service-ids-check.sh"
```

sample-test-from-repo

Or the steps-script is part of the git repo as well:

```
metadata:
    format: Lava-Test Test Definition 1.0
    name: sample-test-script
    description: "Sample with script"
    maintainer:
        - your.name@isp.net
    os:
        - openembedded
    scope:
        - functional

run:
    steps:
        - "cd common/scripts"
        - "./service-ids-check.sh"
```

Referencing a git repo in lava job

```
- 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
      from: git
      path: test-suites/short-smoke/service-check.yaml
      name: service-check
```

Sample job in lava ...

- https://lava.automotivelinux.org/scheduler/job/918/resubmit

Call to action!

Action Required !

- For your *PLATFORM* component (libraries etc.), make sure there is a Yocto Project compatible ptest
- For your APPLICATION make sure to have a wrapper script for the test on the target available (could be already the case)
- For your APPLICATION: write a testdefinition yaml and upload it to qa-testdefinitions repo

Whats next ?

Next steps

- Work on application test workflow
- Generalize and make part of app templates
- Enable API and coverage tests

- Join the CIAT calls an tuesdays to discuss platform and app testing further

QA

Thank you.

Contact:

jsmoeller@linuxfoundation.org

References

- 2017 AMM Talk on writing new tests: http://bit.ly/2Il5SVy
- ptest: https://wiki.yoctoproject.org/wiki/Ptest
- gcov wip: http://bit.ly/2M4CWMQ
- Writing tests for lava: http://bit.ly/2ywcDgQ