Automating Tests With LAVA and BATS

Unit and Integration Testing to avoid Regressions

Diogo Mattioli & Lukas Rabener

INNOMOTICS

Who we are

Diogo Mattioli

Software Engineer

https://linkedin.com/in/diogomattioli

Lukas Rabener

Software Architect and Developer

https://de.linkedin.com/in/lukas-rabener-1ab090193

Agenda

1. Introduction
2. LAVA
3. Testing without a testing framework
4. BATS framework
5. BATS and LAVA integration
6. Running tests outside the pipeline
7. Integration testing
8. Unit testing + Refactoring scripts

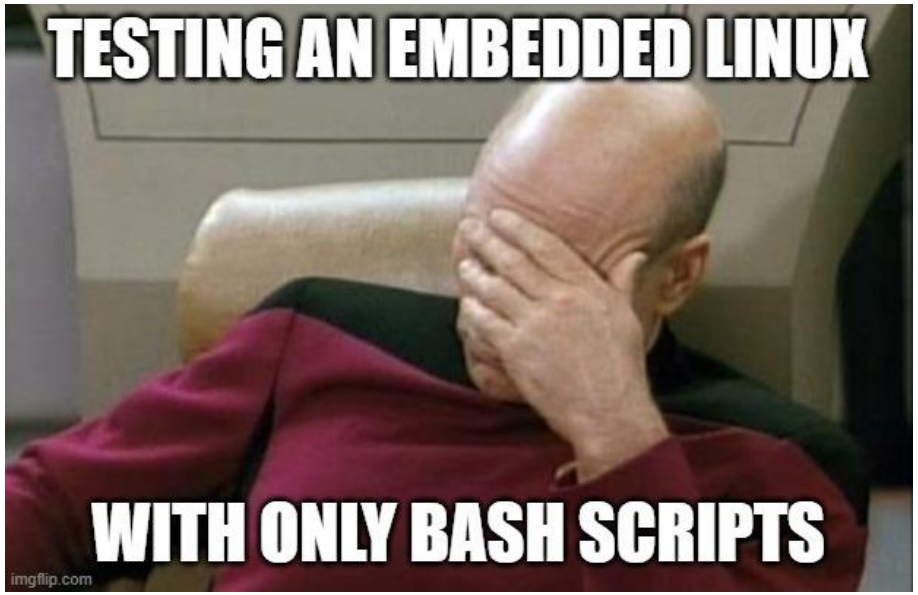
INNOMOTICS

Introduction

How to make test automation easy and scalable on embedded Linux?

We want:

- simple & clear test structure
- support unit & integration tests in bash
- automated in CI/CD and available during development running the same commands
- running virtualized or on hardware



INNOMOTICS

LAVA

- We use LAVA to deploy onto physical and virtual hardware
- But what about the Tests itself?

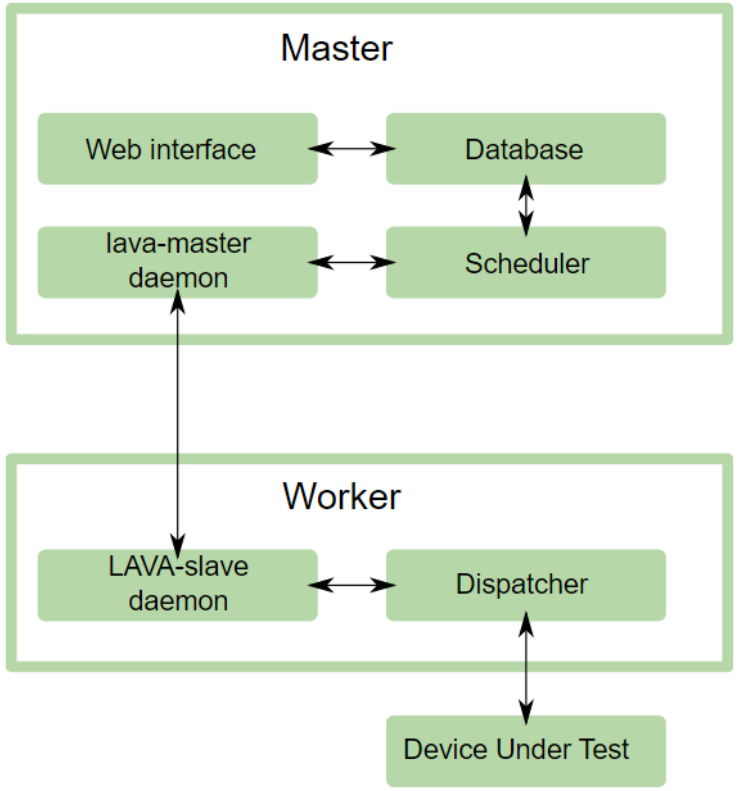
-> LAVA is not a Test Framework

What is LAVA?

- LAVA is the Linaro Automation and Validation Architecture.
- LAVA is a continuous integration system for deploying operating systems onto physical and virtual hardware for running tests. Tests can be simple boot testing, bootloader testing and system level testing, although extra hardware may be required for some system tests. Results are tracked over time and data can be exported for further analysis.

What is LAVA **not**?

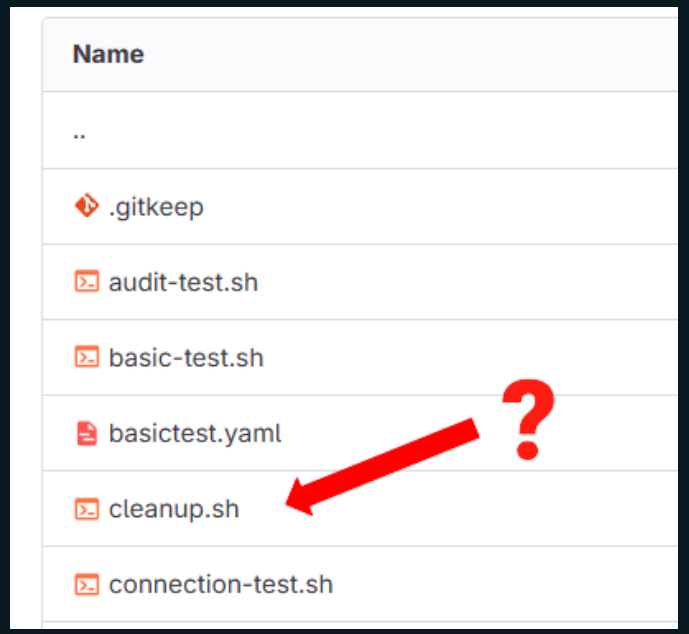
- LAVA is **not** a set of tests - it is infrastructure to enable users to run their own tests. LAVA concentrates on providing a range of deployment methods and a range of boot methods. Once the login is complete, the test consists of whatever scripts the test writer chooses to execute in that environment.



INNOMOTICS

The Problem

Testing without a testing framework creates to much Boilerplate




```
# Check if the service is running
if sudo systemctl is-active --quiet $service_name.service; then
    # Check if the service is listening on the given port
    if sudo ss -ltnp | grep ":$port" | grep -qi "$service_name"; then
        echo "$service_name is running and listening on port $port."
        lava-test-case $service_testid-check-$service_name-listening-port$port --result pass
    else
        echo "$service_name is running but not listening on port $port."
        lava-test-case $service_testid-check-$service_name-listening-port$port --result fail
        teststatus=1
    fi
else
    echo "$service_name is not running."
    lava-test-case $service_testid-check-$service_name-listening-port$port --result fail
    teststatus=1
fi
```


INNOMOTICS

The Solution

We need a bash testing framework emitting signals which Lava can understand

➔ **Bats-core: Bash Automated Testing System**

```
<LAVA_SIGNAL_TESTSET START nginx/nginx.bats>
Received signal: <TESTSET> START nginx/nginx.bats
Starting test_set nginx/nginx.bats
<LAVA_SIGNAL_TESTCASE TEST_CASE_ID=nginx_service_is_running RESULT=pass>
Received signal: <TESTCASE> TEST_CASE_ID=nginx_service_is_running RESULT=pass
case: nginx_service_is_running
definition: 0_bats-core-test
endtc: 1569
result: pass
set: nginx/nginx.bats
starttc: 1569
<LAVA_SIGNAL_TESTCASE TEST_CASE_ID=nginx_service_is_enabled RESULT=pass>
Received signal: <TESTCASE> TEST_CASE_ID=nginx_service_is_enabled RESULT=pass
case: nginx_service_is_enabled
definition: 0_bats-core-test
endtc: 1572
result: pass
set: nginx/nginx.bats
starttc: 1572
<LAVA_SIGNAL_TESTCASE TEST_CASE_ID=nginx_is_running_and_listening_on_port_80_http RESULT=pass>
Received signal: <TESTCASE> TEST_CASE_ID=nginx_is_running_and_listening_on_port_80_http RESULT=pass
case: nginx_is_running_and_listening_on_port_80_http
definition: 0_bats-core-test
endtc: 1575
result: pass
set: nginx/nginx.bats
starttc: 1575
```

Name ↓↑	Test Set ↓↑	Result ↓↑
nginx_service_is_running	nginx/nginx.bats	✔ pass
nginx_service_is_enabled	nginx/nginx.bats	✔ pass
nginx_is_running_and_listening_on_port_80_http	nginx/nginx.bats	✔ pass

References:

- https://github.com/bats-core/bats-core
- https://events19.linuxfoundation.org/Teaching-your-Test-Framework-to-Speak-LAVA

INNOMOTICS

BATS Framework

Bash Automated Testing System

- TAP (Test Anything Protocol)
- Setup and Teardown
- Tracing
- Verbosity
- RUN helper

```bash
#!/usr/bin/env bats

setup_file() {
    echo "executed BEFORE all the tests in this file"
}

teardown_file() {
    echo "executed AFTER all the tests in this file"
}

setup() {
    echo "executed BEFORE every single test in this file"
}

teardown() {
    echo "executed AFTER every single test in this file"
}

@test "Test that checks if file exists with RUN" {
    run ls /var/mutable-data
    [[ "$status" -eq 0 && "$output" == *"myfile"* ]]
}

@test "Test that checks if file exists with RUN expecting status code 0" {
    run -0 ls /var/mutable-data
    [[ "$output" == *"myfile"* ]]
}

@test "Test that checks if file exists without RUN" {
    ls /var/mutable-data/myfile
}
```


INNOMOTICS

BATS Framework

```
# bats dummy.bats
dummy.bats
 ✓ Test case that checks if file exists with RUN
 ✓ Test case that checks if file exists with RUN expecting status code 0
 ✓ Test case that checks if file exists without RUN

3 tests, 0 failures
```

```
# bats dummy.bats
dummy.bats
 ✗ Test case that checks if file exists with RUN
   (in test file dummy.bats, line 16)
     `[[ "$status" -eq 0 && "$output" == *"myfile"* ]]' failed
   executed BEFORE every single test in this file
   executed AFTER every single test in this file
 ✗ Test case that checks if file exists with RUN expecting status code 0
   (in test file dummy.bats, line 20)
     `[[ "$output" == *"myfile"* ]]' failed
   executed BEFORE every single test in this file
   executed AFTER every single test in this file
 ✗ Test case that checks if file exists without RUN
   (in test file dummy.bats, line 23)
     `ls /var/mutable-data/myfile' failed with status 2
   executed BEFORE every single test in this file
   ls: cannot access '/var/mutable-data/myfile': No such file or directory
   executed AFTER every single test in this file

3 tests, 3 failures
```


INNOMOTICS

BATS and LAVA integration

Test Anything Protocol
https://testanything.org/

“A simple text-based interface between testing modules in a test harness.

It decouples the reporting of errors from the presentation of the reports.“

```bash
#!/usr/bin/env bash
set -e
trap '' INT

source "/usr/lib/bats-core/formatter.bash"

...

bats_tap_stream_ok() { # [<test index> <test name>
    printf "<LAVA_SIGNAL_TESTCASE TEST_CASE_ID=%s RESULT=pass>\n" "${2// /_}"
}

bats_tap_stream_not_ok() { # <test index> <test name>
    printf "<LAVA_SIGNAL_TESTCASE TEST_CASE_ID=%s RESULT=fail>\n" "${2// /_}"
}

bats_tap_stream_skipped() { # <test index> <test name> <reason>
    printf "<LAVA_SIGNAL_TESTCASE TEST_CASE_ID=%s RESULT=skip>\n" "${2// /_}"
}

bats_tap_stream_comment() { # <comment text without leading '# '>
   printf "# %s\n" "$1"
}

...
```


INNOMOTICS

BATS and LAVA integration

```
# bats --formatter $(pwd)/tap-lava dummy.bats
<LAVA_SIGNAL_TESTCASE TEST_CASE_ID=Test_case_that_checks_if_file_exists_with_RUN RESULT=pass>
<LAVA_SIGNAL_TESTCASE TEST_CASE_ID=Test_case_that_checks_if_file_exists_with_RUN_expecting_status_code_0 RESULT=pass>
<LAVA_SIGNAL_TESTCASE TEST_CASE_ID=Test_case_that_checks_if_file_exists_without_RUN RESULT=pass>
```

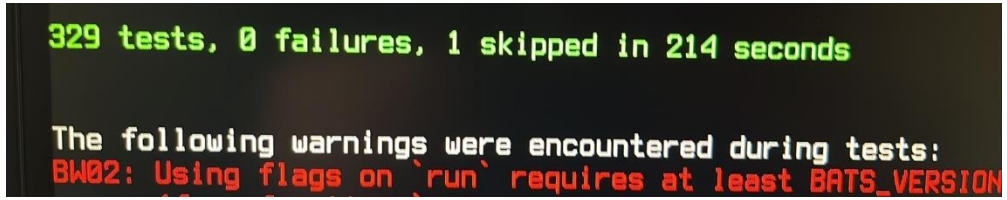
```
# bats --formatter $(pwd)/tap-lava dummy.bats
<LAVA_SIGNAL_TESTCASE TEST_CASE_ID=Test_case_that_checks_if_file_exists_with_RUN RESULT=fail>
# (in test file dummy.bats, line 16)
#   `[[ "$status" -eq 0 && "$output" == *"myfile"* ]]' failed
# executed BEFORE every single test in this file
# executed AFTER every single test in this file
<LAVA_SIGNAL_TESTCASE TEST_CASE_ID=Test_case_that_checks_if_file_exists_with_RUN_expecting_status_code_0 RESULT=fail>
# (in test file dummy.bats, line 20)
#   `[[ "$output" == *"myfile"* ]]' failed
# executed BEFORE every single test in this file
# executed AFTER every single test in this file
<LAVA_SIGNAL_TESTCASE TEST_CASE_ID=Test_case_that_checks_if_file_exists_without_RUN RESULT=fail>
# (in test file dummy.bats, line 23)
#   `ls /var/mutable-data/myfile' failed with status 2
# executed BEFORE every single test in this file
# ls: cannot access '/var/mutable-data/myfile': No such file or directory
# executed AFTER every single test in this file
```


INNOMOTICS

Running tests outside the pipeline

- before:
  - tests cloned by Lava job & executed
- now:
  - including tests into rootfs
  - test recipe included in dev-image

- Enables Testing during development
  ➔ run tests with a single CMD
  ➔ qemu, VM or on hardware




```
inherit dpkg-raw

DESCRIPTION = "Testing files"
DEBIAN_DEPENDS += ”bats, dnsutils, curl, sntp"

FILESEXTRAPATHS:prepend := "${TOPDIR}/../:"

SRC_URI = " \
    file://tests/ \
    "

do_install[cleandirs] += "\
    ${D}/tests \
    "

do_install() {
    cp -rf ${WORKDIR}/tests/* ${D}/tests/
}
```


INNOMOTICS

Integration testing (System testing)

- Verify that build was correct
- Check services
- Config change during runtime
- Disk tests
- Connectivity tests
- Hardware tests

INNOMOTICS

Integration testing (System testing)

```
#!/usr/bin/env bats

@test "/ filesystem type is squashfs" {
    run -0 findmnt -no FSTYPE /
    [[ "${output}" == squashfs ]]
}

@test "/var filesystem type is ext4" {
    run -0 findmnt -no FSTYPE /var
    [[ "${output}" == ext4 ]]
}

@test "/var is writable" {
    run -0 mktemp -p /var
    run -0 rm -f $output
}

@test "/tmp is writable" {
    run -0 mktemp
    run -0 rm -f $output
}

@test "/etc is read only" {
    run ! mktemp -p /etc
}
```

```
#!/usr/bin/env bats

@test "file resolv.conf is present on the filesystem" {
    # in case there was a problem with the build
    [ -f /etc/resolv.conf ]
}

@test "file resolv.conf must not be empty" {
    # in case another process is overwriting it
    [ -s /etc/resolv.conf ]
}

@test "nameserver in resolv.conf for cloudflare ip address exists" {
    grep -q "nameserver 1.1.1.1" /etc/resolv.conf
}

@test "nameserver in resolv.conf for google dns ip address exists" {
    grep -q "nameserver 8.8.8.8" /etc/resolv.conf
}

@test "dns resolution for unprivileged user fails" {
    # 9: No reply from server
    run -9 dig www.google.com +time=1 +tries=1
}
```


INNOMOTICS

Unit testing + Refactoring scripts

- “Unit testing involves isolating units so that functionality can be confirmed before units are integrated with other parts of the application”
  https://www.ibm.com/think/topics/unit-testing

- You decide what is a unit

- Refactor: separate the units in your scripts and test them individually

INNOMOTICS

Unit testing + Refactoring scr

```bash
#!/bin/bash

# Simulated file operations
echo "Creating temporary files..."
touch temp_file_1.txt
touch temp_file_2.txt
echo "Files created."

# Simulated cleanup
echo "Cleaning up..."
rm temp_file_*.txt
echo "Cleanup complete."
```

```bash
#!/bin/bash

readonly FILE_1_PATH="temp_file_1.txt"
readonly FILE_2_PATH="temp_file_2.txt"

create_temporary_files() {
    local dir=$1
    local file1=$2
    local file2=$3
    echo "Creating temporary files..."
    touch "${dir}/${file1}"
    touch "${dir}/${file2}"
    echo "Files created."
}

cleanup() {
    local dir=$1
    echo "Cleaning up..."
    rm "${dir}/*"
    echo "Cleanup complete."
}

main() {
    create_temporary_files "/tmp" ${FILE_1_PATH} ${FILE_2_PATH}
    cleanup "/tmp"
}

# Script Entry Point
if [[ "${BASH_SOURCE[0]}" == "$0" ]]; then
    main
fi
```

scripts

```bash
#!/bin/bash

readonly FILE_1_PATH="temp_file_1.txt"
readonly FILE_2_PATH="temp_file_2.txt"

create_temporary_files() {
    local dir=$1
    local file1=$2
    local file2=$3
    echo "Creating temporary files..."
    touch "${dir}/${file1}"
    touch "${dir}/${file2}"
    echo "Files created."
}

cleanup() {
    local dir=$1
    echo "Cleaning up..."
    rm "${dir}/*"
    echo "Cleanup complete."
}

main() {
    create_temporary_files "/tmp" ${FILE_1_PATH} ${FILE_2_PATH}
    cleanup "/tmp"
}

# Script Entry Point
if [[ "${BASH_SOURCE[0]}" == "$0" ]]; then
    main
fi
```

```bash
#!/usr/bin/env bats

source /path/to/script

setup() {
    dir=$(mktemp -d)
}

teardown() {
    rm -rf ${dir}
}

@test "temporary files were created" {
    run -0 create_temporary_files "${dir}" "test1" "test2"
    [ -f "/${dir}/test1" ]
    [ -f "/${dir}/test2" ]
}

@test "temporary files were cleaned up" {
    run -0 touch "${dir}/test1" "${dir}/test2"
    run -0 cleanup "${dir}"
    [ ! -f "/${dir}/test1" ]
    [ ! -f "/${dir}/test2" ]
}
```


INNOMOTICS

Thank you!

Questions?

INNOMOTICS