Continuous testing in a cloud based infrastructure using virtualization and real hardware in the loop

Armand Bénéteau – FOSDEM – February 5th 2022
Agenda

- Introduction and context
- Q.A. goals
- RTM introduction and overview
- RTM in the Community stack
- Real RTM challenges
- LAVA integration in real RTM
- Board integration in real RTM
- Test reporting in Q.A. system
- Roadmap
- Conclusion
About me

• Software and QA engineer @ IoT.bzh
• Formerly:
  - Master degrees in IT/electronic engineering from:
    • INSA (Rennes, France)
    • Strathclyde University (Glasgow, Scotland)
  - Low Power IOT (LoRaWAN technology)
    • Wireless sensors firmware (C)
    • LoRaWAN protocol implementation (C)
    • LoRaWAN gateway, running on Linux (C)
• armand.beneteau@iot.bzh
• https://www.linkedin.com/in/armand-beneteau/
IoT.bzh at a glance

Our location
Brittany

Our 30-year OS background

Our team
~30 engineers

Our location
Brittany

Our 30-year OS background

Our team
~30 engineers

European CyberSecurity Organisation Cyber Valleys mapping

Our location
Brittany

Our 30-year OS background

Our team
~30 engineers

Our location
Brittany

Our 30-year OS background

Our team
~30 engineers

Our product
redpesk® is a pre-integrated « ready-to-use » CI/CD SW factory generating a custom & secured OS, with Long Term Support
Introduction and Context

• **Code complexity** is increasing in systems
  Boeing 787 (2004): 14 million lines of code[^1]
  Average high-end car (2012): 100 million lines of code[^2]

• **L.T.S.** is mandatory for industrial systems
  Average age of european car: 12 years old (source: ACEA)
  Approximately the same in the U.S. (source: IHS Market)

• **Cybersecurity** is not an option anymore
  In the first 6 months of 2021: > 1.5 billion attacks on IoT devices (source: Kaspersky)

• **Conclusion: Automatic CI/CT/QA infrastructure is a must have**

[^2]: https://www.wired.com/2012/12/automotive-os-war/
QA in the factory

Native BSPs
- yocto
- ST
- Qualcomm
- NXP
- Intel

BSPs
- Redpesk Fontend Web/Cli
- Redpesk API
- yocto bitbake
- Koji build system
- Kofishop
- License
- Vulnerability scanner
- CI/CD/QA
- Gitpkgd
- Koji
- Yocto
- Security
- SDK
- OTA
- Release Mgmt
- App/User-code
- Middlewares
- Upstream
- Linux mainline
- CentOS
- Debian
- Fedora
- Github
- Gitlab
- Upstream
- Linux mainline
- ST
- Qualcomm
- NXP
- Intel

Redpesk Factory

C.T. in cloud using virtualization and real hardware
Q.A. goals: workflow

1. Checks out code
2. Commits changes
3. CI Server initiates a build
4a. Build Passed
4b. Build Failed
5. Unit test
6. Real Integration test

Users notification: Fail or Pass

Agile Manager
Agile Developer #1
Agile Developer #2

Source Control Server

Build

Virtual Integration test
Q.A. goals: reporting
RTM introduction

- **Integration tests** run in our **R.T.M.: Rackable Test Modules**

RTM are our solution to run integration tests within the redpesk infrastructure.

They are intended to meet user requirements regarding qualifications, certifications, continuous integration, etc.

They can be dynamically started by the infrastructure or by the developer to run integration tests.

They are the heart of the continuous integration and continuous testing inside redpesk factory.
RTM overview: infrastructure overall view

Client PC

WebUI or rp-cli

VPN Client

Internet access through client machine

Cloud

VPN Tunnel

Virtual

redpesk® factory

redpesk® Repositories

client packages download

redpesk image download

RTMSHIP

Real

redpesk® Backend

Virtual

RTM 1

RTM 2

RTM 3

Virtual

OR

Real

VPN Tunnel

FIREWALL

FIREWALL

FIREWALL

Virtual

LXC

LXC

Routing Table

QEMU redpesk x86_64 or aarch64

Dispatcher

QEMU Controller

Routing

Table

Video

Audio

IO

USB

Power supply & Monitoring

(A, W, °C)

Target Board

Master Board

Routing Table

Dispatcher

Hardware RTM

Hardware RTM

Hardware RTM

VPN Server

VPN

Controller

QEMU

Routing Table

Dispatcher

Power supply

(V CC)

Audio

Video

CAN

UART

VCC

E32

Target Board

FIREWALL

FIREWALL

FIREWALL
Developer testing process in Community

Definition of “redtests”[1]:
- Allows one to have a “standard” for tests in CI

Really simple:
- A “redtest” subpackage
- ‘run-redtest’ script in this package
- Test logs need to respect “TAP” format[2]

---

RTM infrastructure in Community

Client PC

WebUI
or
rp-cli

Cloud

redpesk® factory

LXC
LXC

QEMU redpesk
x86_64 or aarch64

QEMU Controller

RTMShip

redpesk® Repositories

redpesk® Backend

IOT BZH
C.T. in cloud using virtualization and real hardware

FOSDEM - 05-02-2022
Demo in redpesk “Community”
Real RTM challenges

➢ **Sharing boards** between users
  - *We do not have one board per user (price, availability)*
  - *The real RTM system needs to manage the users access*

➢ Management of **board’s power supply**
  - In order to start and stop the boards correctly

➢ Management of **redpesk OS image loading**
  - We need to load a full redpesk image (several partitions and size > 2G)

➢ Management of **board’s boot**
  - Grub, uboot, prompt, etc.
Real RTM: LAVA, the “missing link”

- **LAVA** *(Linaro Automated Validation Architecture)*[^1]:
  - Continuous Integration system for deploying OS onto physical and virtual hardware for running tests
  - Used a lot in Kernel validation (e.g.: in KernelCI[^2])
  - Fully open-source[^3]
  - Already existing board definitions
  - Uboot, grub and fastboot management

---

[^1]: https://www.lavasoftware.org/
[^2]: https://kernelci.org/
[^3]: https://git.lavasoftware.org/lava
Real RTM integration in redpesk CI

- **rtm-farm-client**
  - New async µservice
  - Xmlrpc and zmq to communicate with LAVA master
  - Allows homogeneous communication with backend (rest API + ws)

- Some refactoring in backend needed

- Integration of “redpesk boards” in LAVA environment
Real RTM architecture

➢ 1 dispatcher per DUT: to be able to run more complex tests
  - HDMI grab
  - Audio grab
  - Etc…

➢ Job submissions, test results, test logs fully managed by LAVA

➢ LAVA master – dispatcher communication is LAVA business not redpesk
Powering a real RTM

How to control the power supply?

- First easy solution: a remotely controlled multi-socket
  - *Energenie EG-PMS2-LAN*
  - Controlled through ethernet
  - Binary available in Linux distributions\(^1\): `/usr/bin/egctl`

- Second, more complete, solution: laboratory power supply
  - *Joy-it JT-DPM8605*
  - Controlled through RS485 (ModBus) protocol
  - Simulate “low” or “high” voltage situations to test boards in their limits

\(^1\) [https://manpages.ubuntu.com/manpages/focal/man1/egctl.1.html](https://manpages.ubuntu.com/manpages/focal/man1/egctl.1.html)
Real RTM: network boot (with PXE)

➢ *First option: NFS filesystem*
  - Not usable for us because **NFS does not propagate the SELinux or Smack labels**…

➢ *Second option: NBD (Network Block Devices) filesystem*
  - *Works well* (really fast on a good network)
  - No need to flash the redpesk image locally → **does not wear the board memory out** (SD card, eMMC)
  - But:
    • Slower or faster → **not exactly the same behaviour** as local boot
Real RTM: boot with fastboot

➢ **Fastboot** *(coming from Android)*

- The DUT behaves as **USB storage** where the image can be **flashed**
- To enter in “fastboot” mode, “U-boot” needs to be stopped
- **Supported by LAVA**
- Allows us to integrate 3 boards out of 4:
  - **Solidrun Solidsense**
  - **Renesas-Gen3**
  - **Raspberry Pi 4**
Real RTM: boot on USB

➢ If fastboot not supported: **USB gadget**
  - USB gadget enabled in dispatcher (needs USB OTG)
  - Dispatcher behaves as **USB storage** → **DUT boot on USB** in this case
  - No need to flash the boards eMMC → does not wear the board memory out (SD card, eMMC)
  - In the future, it will allow us to simulate USB devices
    • Mouse
    • Keyboards
    • Etc.
  - Allows us to integrate the last board:
    • **Intel Up-Board**
Test reporting

➢ Once integration tests have passed or failed, **reporting needs to be done on the results**
  
  • Boot logs (if relevant)
  • stdout/stderr outputs during test runs
  • “.tap” file containing the test results

➢ If the tests are **successful** (virtual and real), the package can go to the **next step**
  
  • Vulnerability scanner
  • Licence analysis
  • Etc.
Test reporting in Community

C.I. Test

<table>
<thead>
<tr>
<th>General Info</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Test ID</td>
<td>26</td>
</tr>
<tr>
<td>Application Name</td>
<td>helloworld-binding</td>
</tr>
<tr>
<td>Status</td>
<td>success</td>
</tr>
<tr>
<td>Created at</td>
<td>Nov 23, 2021, 11:14:27 AM</td>
</tr>
<tr>
<td>Ended at</td>
<td>Nov 23, 2021, 11:15:35 AM</td>
</tr>
<tr>
<td>Board model</td>
<td></td>
</tr>
<tr>
<td>Error message</td>
<td></td>
</tr>
<tr>
<td>Total tests run</td>
<td>10</td>
</tr>
<tr>
<td>Succeeded tests</td>
<td>9</td>
</tr>
<tr>
<td>Failed tests</td>
<td>0</td>
</tr>
<tr>
<td>Skipped tests</td>
<td>1</td>
</tr>
<tr>
<td>Log file</td>
<td>tests_logs.zip</td>
</tr>
<tr>
<td>Tap file</td>
<td>helloworld-binding_all_tests.tap</td>
</tr>
</tbody>
</table>

Test result file: helloworld-binding_all_tests.tap

```plaintext
1 # Filename: helloworld.tap
2 1..5
3 # Started on Tue Nov 23 11:15:29 2021
4 # Starting class: testPingSuccess
5      Begin testPingSuccess
6      End testPingSuccess
7 ok 1 testPingSuccess.testFunction
8 # Starting class: testPingSuccessAndResponse
9 ok 2 testPingSuccessAndResponse.testFunction
10 # Starting class: testPingSuccessCallback
11 ok 3 testPingSuccessCallback.testFunction
12 # Starting class: testPingError
13 ok 4 testPingError.testFunction
14 # Starting class: testPingErrorAndResponse
15 ok 5 testPingErrorAndResponse.testFunction
16 # Run 5 tests in 0.001 seconds, 5 successes, 0 failures
17 # Filename: mpi_tests.tap
18 1..5
19 # Started on Tue Nov 23 11:15:29 2021
20 # Starting class: TestListSuccess
21 ok 1 TestListSuccess.testFunction
22 # Starting class: TestSubscribeSuccess
23 ok 2 TestSubscribeSuccess.testFunction
24 # Starting class: TestUnsubscribeSuccess
25 ok 3 TestUnsubscribeSuccess.testFunction
26 # Starting class: TestWrongVerbError
27 ok 4 TestWrongVerbError.testFunction
```
Roadmap

➢ Advanced test implementation in LAVA – redpesk CI
  ✔ Tests that need external processes (HDMI or audio grab, etc.)
  ✔ These processes need to be run on the “dispatcher”
➢ Remote access to boards in “development mode”
  ✔ Directly through VPN
  ✔ In LAVA, it corresponds to the “hacking session”[1]
➢ Adding test libraries shared between projects
➢ Integration of external module in order to go further in QA
  ✔ Code scanner for cybersecurity
  ✔ Flowchart generator (certification)
  ✔ Etc.

[1] https://docs.lavasoftware.org/lava/hacking-session.html
Conclusion

➢ Continuous testing is a must have because of:
   ✔ Increasing code complexity
   ✔ L.T.S. need for industrial systems
   ✔ Cybersecurity concern

➢ Both virtual and real boards must be in the C.I. loop
   ✔ For a lot of tests a virtual target is enough

➢ Iot.bzh answers
   ✔ Virtual RTMs (Qemu in LXC) → Available in Community
   ✔ Real RTMs → Work in progress!

➢ Continuous testing is a part of the QA system
   ✔ Can be completed with other external modules
Links

- **redpesk**
  - Website: https://redpesk.bzh/
  - Documentation: https://docs.redpesk.bzh/
  - Sources: https://github.com/redpesk/readme

- **IoT.bzh**
  - Website: https://iot.bzh/
  - Publications: https://iot.bzh/en/publications
  - Videos: https://vimeo.com/search?q=redpesk

- **Community Support**
  - Matrix.org: +redpesk:matrix.org