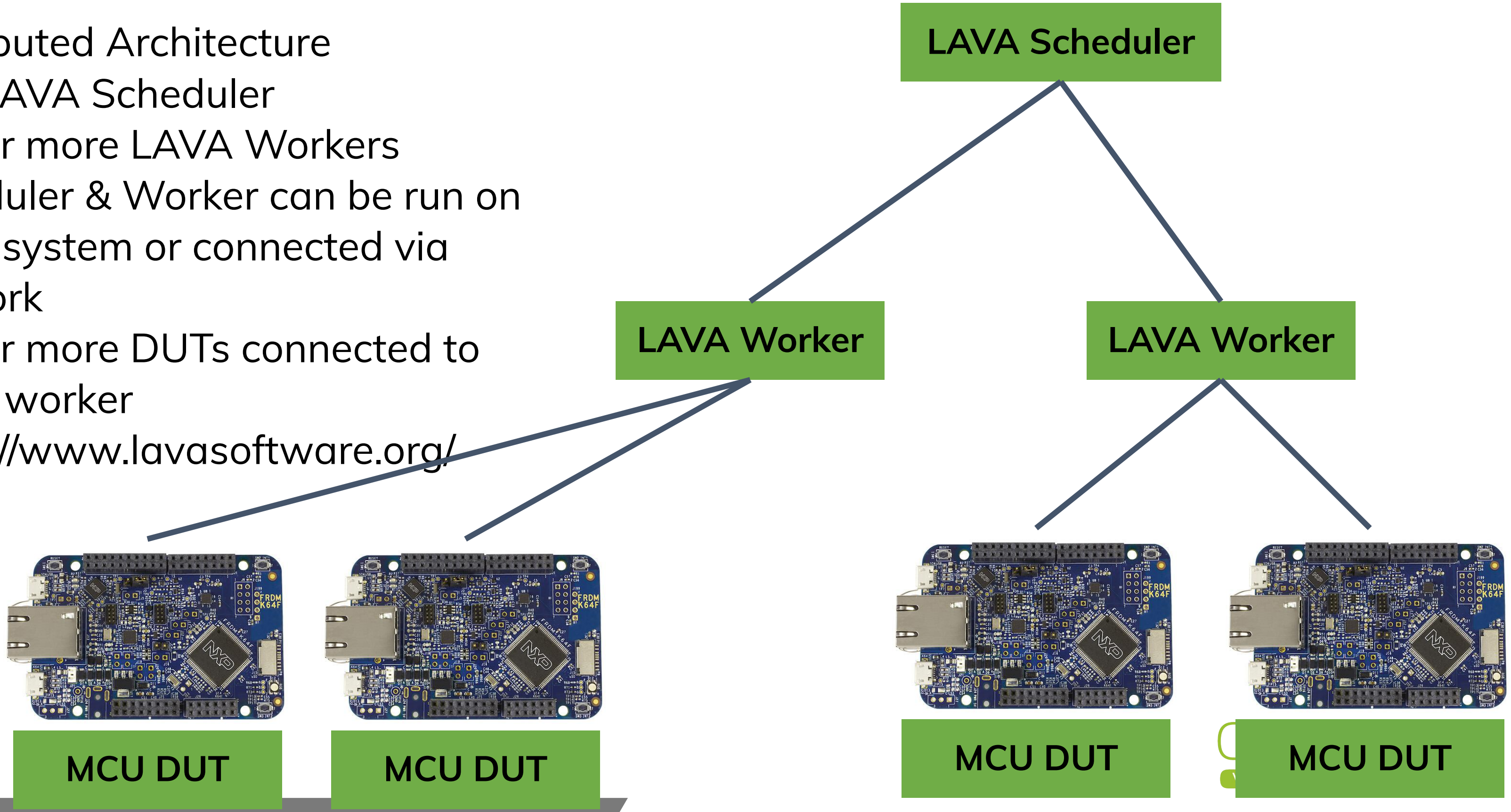
LVC20-310 Testing IoT Devices: Design and Progress from LITE Team

Kumar Gala & Paul Sokolovsky

Linaro Connect VIRTUAL 2020

LAVA Architecture

- Distributed Architecture
- One LAVA Scheduler
- One or more LAVA Workers
- Scheduler & Worker can be run on same system or connected via network
- One or more DUTs connected to LAVA worker
- https://www.lavasoftware.org/

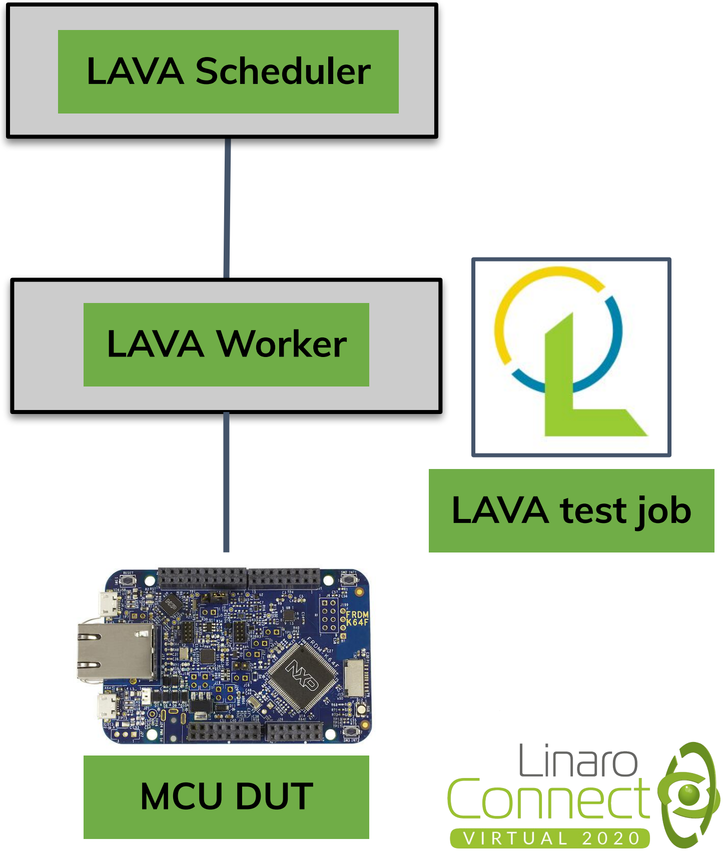


MCU testing with LAVA

- Basic MCU testing utilizes flashing a test image to DUT and verify output of test on UART for success / failure
- Requires ability to flash the DUT which may require DUT specific tooling. Some common tooling exists via pyOCD, OpenOCD, Segger J-Link and USB mass storage, however some board still require either specific versions of these tools or board/SoC specific tooling
- Due to the extremely low cost of the DUTs is extremely easy for every developer to have a DUT on their “desk”
- Therefore having the same workflow for developer “desk” and testing CI loop is highly desirable to reduce cost and effort in both developing tests and debugging test failures

Lava MCU Testing (today)

- Utilize LAVA test job description to specify “boot method” and “test monitors”
- “boot method” used to flash test image to board
    - Pyocd, openocd, jlink, and usb mass storage supported by LAVA for flashing
    - Need LAVA awareness of flashing tool
    - Tools need to be installed on LAVA worker, possible version issues
- “test monitors” used to specify expected output result of test
- Requires each test to have a corresponding LAVA test job definition
- Changes to tests (modifications, removal, additions) have to be kept in sync
- Failure reproduction on developer system requires different workflow

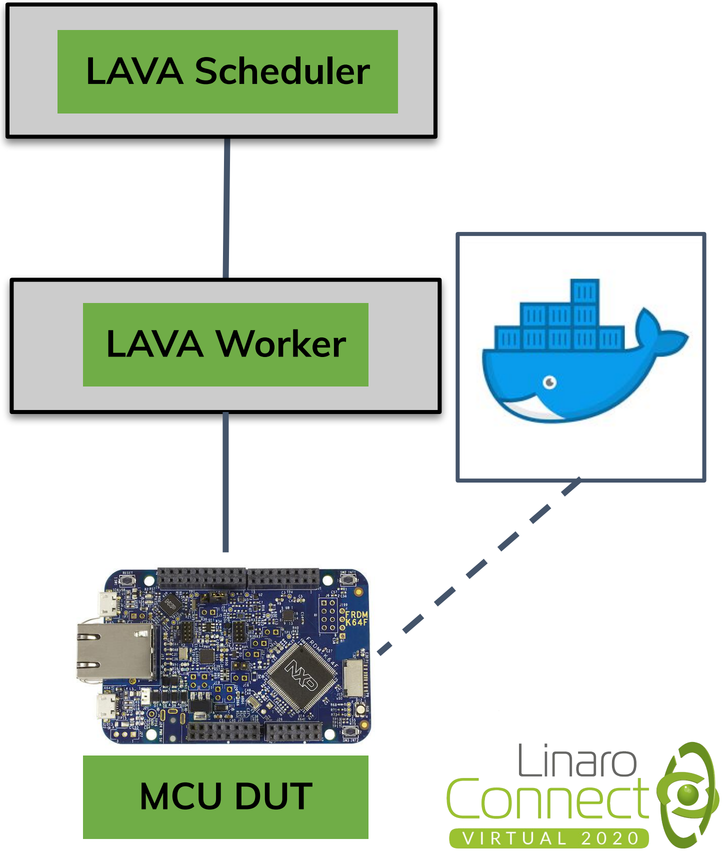


Zephyr testing with LAVA (today)

- Zephyr provides a developer oriented testing infrastructure centered around a python tool called `sanitycheck`
- `sanitycheck` tool is able to both build tests for a given board (DUT) and run various tests based on test needs on the board.
- Zephyr provides infrastructure around the board to know how to flash images to the board
- Current Linaro testing of Zephyr utilizes `sanitycheck` in a CI flow to create and publish testing artifacts
- LAVA is utilized to deploy (flash), record, and report the results of each runnable Zephyr test
- A python script exists to convert Zephyr test metadata to LAVA job definition files.
- Script has to be kept in sync with Zephyr testing changes.

Lava MCU Testing (docker)

- Utilize LAVA docker shell container for test job
- USB connection and devices for debug/flash passed through to Container
- Container provides software tools to connect and flash the board (able to easily handle unique software versions or tools for flashing)
- Able to utilize same test software infrastructure that a developer utilizes on their “desk”
- Easy to reproduce failures locally due to same testing software
- LAVA utilized to dispatch tests & record results
- Testing software can be “lava-ized” by simple LAVA specific console output to denote start/end and pass/fail of a test
- Fairly easy to adapt to any testing infrastructure provided by software under test



Zephyr testing with LAVA and docker

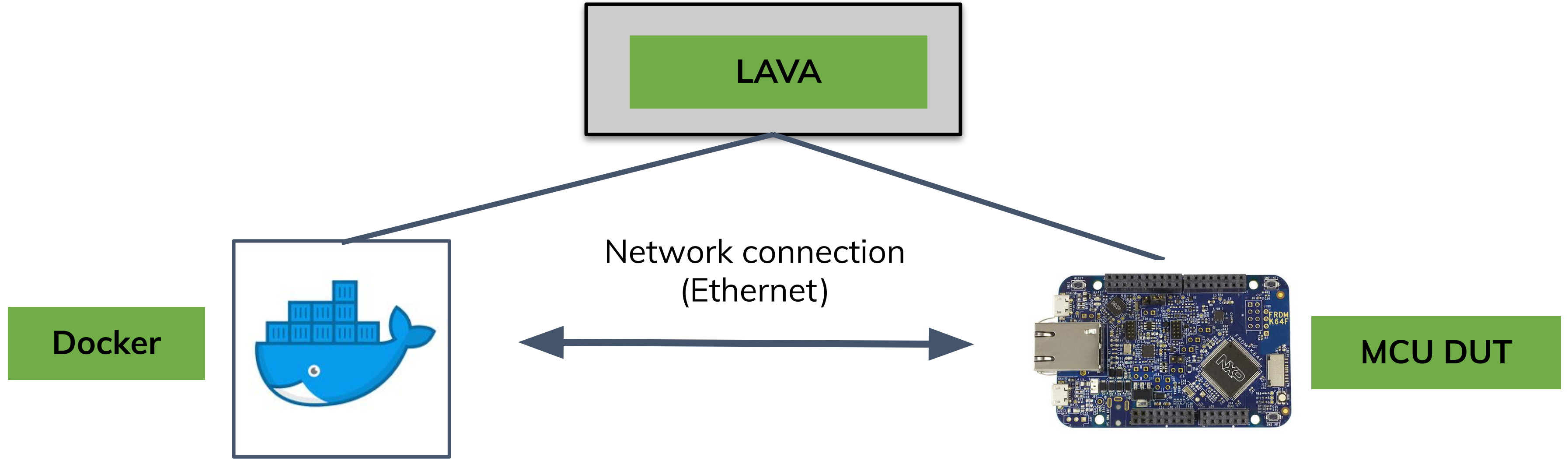
- Provide a docker container that is able to run Zephyr sanitycheck
- Requires ability to flash the DUT which may require DUT specific tooling. Some common tooling exists via pyOCD, OpenOCD, Segger J-Link and USB mass storage, however some board still require either specific versions of these tools or board/SoC specific tooling
- Due to the extremely low cost of the DUTs is extremely easy for every developer to have a DUT on their “desk”
- Therefore having the same workflow for developer “desk” and testing CI loop is highly desirable to reduce cost and effort in both developing tests and debugging test failures

DEMO

Zephyr networking testing in LAVA

- Network testing requires (at least) two devices to test connectivity between. Supported by LAVA as “multinode testing”.
- One node is a Zephyr device
- For another node, just use “a virtual device”, a Docker container running arbitrary Linux software to test connectivity to/from Zephyr.
- “`test: interactive`” native LAVA test method, supporting detailed output matching and ability to inject input.
- Challenge: no multinode synchronization support for “`test: interactive`”.
- Was implemented in LAVA 2020.05, with further fixes in 2020.08.

Setup for networking LAVA test



Runs arbitrary (Linux) client/server to connect to / be connected from DUT.

Runs a client/server Zephyr application

Zephyr networking testing: current results

- Running as part of the CI loop:
  - ICMP tests (ping)
  - Zephyr HTTP server test
- Zephyr HTTP client test is available for local LAVA setup, but currently doesn't work in LAVA Lab (apparently, due to heavy firewall setup there).
- CI tests were used to actually catch issues in the new "TCP2" stack while preparing for Zephyr 2.4 release.

- Reports are available at https://qa-reports.linaro.org/lite/zephyr-net/

Future plans:

- More networking tests can/should be written: TLS, MQTT, LwM2M, etc.

DEMO - Backup Material

LAVA docker job definition

```
# Zephyr JOB definition for frdm-k64f
device_type: 'frdm-k64f'
job_name: 'zephyr docker'
timeouts:
  job:
    minutes: 3
  action:
    minutes: 3
priority: medium
visibility: public
actions:
- test:
    docker:
        image: kumargala/ci-run
    definitions:
    - repository:
        metadata:
          format: Lava-Test Test Definition 1.0
          name: foo bar
        run:
          steps:
          - wget -q https://people.linaro.org/~kumar.gala/zephyr-sanitycheck-hw.tar.gz
          - tar xf zephyr-sanitycheck-hw.tar.gz
          - cd zephyr
          - west init -l
          - west config zephyr.base zephyr
          - ./scripts/sanitycheck -v -i -p frdm_k64f --device-testing --device-serial /dev/ttyACM2 --test-only --west-flash
      from: inline
      name: bar
      path: foobar
    timeout:
      minutes: 2
```

Demo: docker container - kumar/ci-run

- Container based on ubuntu:18.04
- Pulls in basic Zephyr build and test requirements to be able run `sanitycheck`
- Supports pyOCD and OpenOCD for flashing of boards
- Utilizes Zephyr’s SDK hosttool package for openOCD
- https://github.com/galak/docker-ci-run

Demo: zephyr-sanitycheck-hw.tar.gz

- Pre-package minimal artifacts for device testing with `sanitycheck`
- Built `samples/basic` and `samples/hello_world` from Zephyr for FRDM-K64F
- Includes zephyr `scripts/` directory for both west extensions (for flashing) and `sanitycheck` script
- Includes minimal additional artifacts for tools to run:
  - `boards/arm/frdm_k64f/frdm_k64f.yaml`
  - `samples/basic/*/sample.yaml, samples/hello_world/sample.yaml`
  - `Strip down sanity-out/frdm_k64f/<SAMPLE>/`
    - `zephyr/zephyr.{bin,hex}`
    - `zephyr/runners.yaml, CMakeCache.txt (for west to know how to flash)`

Thank you

Accelerating deployment in the Arm Ecosystem

Linaro Connect VIRTUAL 2020