HKG18 TR12 - LAVA for LITE Platforms and Tests

Bill Fletcher
Training Contents Summary

Part 1
- LAVA overview and test job basics
- Getting started with the Lab Instance
- Anatomy of a test job
- Looking at LAVA results
- Writing tests

Part 2
- LAVA in ci.linaro.org overview
- Invoking LAVA via xmlrpc
- Metadata
- Job templates

- Specifics
  - Material is specific to LITE
  - Emphasis is Zephyr targets rather than Linux (i.e. monolithic images and no shells)

- Out of scope
  - mcuboot*
  - Installing a local LAVA instance
  - Adding new devices
  - Adding new features
  - LAVA/Lab planning

*as far as I can tell mcuboot isn’t supported anywhere yet
LAVA Overview

- The Linaro Automated Validation Architecture
- An automation system for deploying executable images onto physical and virtual hardware for running tests
- Very scalable
- More details at https://validation.linaro.org/static/docs/v2/
- LAVA Lab went live in July 2011 with 2(!) device types
- Features in the latest version:
  - YAML format job submissions
  - Live result reporting
  - A lot of support for scaled and/or distributed instances
Basic Elements of LAVA

- **Web interface** - UI based on the uWSGI application server and the Django web framework. It also provides XML-RPC access and the REST API.
- **Database** - PostgreSQL locally on the master storing jobs and device details
- **Scheduler** - periodically this will scan the database to check for queued test jobs and available test devices, starting jobs on a Worker when the needed resources become available.
- **Lava-master daemon** - This communicates with the worker(s) using ZMQ.
- **Lava-slave daemon** - This receives control messages from the master and sends logs and results back to the master using ZMQ.
- **Dispatcher** - This manages all the operations on the device under test, according to the job submission and device parameters sent by the master.
- **Device Under Test (DUT)**
Dispatchers and Devices

• The picture on the left shows Hikey boards in the Lab connected to one of the Dispatchers

• The Dispatcher in this case provides:
  ○ USB ethernet - Networking
  ○ FTDI serial - console
  ○ USB OTG - interface for fastboot/flashing
  ○ Mode control (via OTG power or not)
  ○ Power control

• The Dispatcher needs to be able to:
  ○ Put the device in a known state
  ○ Deploy the test image to the device
  ○ Boot the device
  ○ Exactly monitor the execution of the test phase
  ○ Put the device back into a known state
LAVA Test Job Basics - a pipeline of Dispatcher actions

- Downloads files required by the job to the dispatcher, to: parameter selects the deployment strategy class
- Boot the device
  - The device may be powered up or reset to provoke the boot.
  - Every boot action must specify a method: which is used to determine how to boot the deployed files on the device.
- Execute the required tests
  - Monitor the test execution
  - Use naming and pattern matching elements to parse the specific test output

- Individual action blocks can be repeated conditionally or unconditionally
- Groups of blocks (e.g. boot, test) can also be repeated
- Other elements/modifiers are: timeouts, protocols, user notifications
A Simplified Example Pipeline of Test Actions

1. deploy
   1.1. strategy class
   1.2. zephyr image url
2. boot
   2.1. specify boot method (e.g. cmsis/pyocd)
3. test
   3.1. monitor patterns

A more complex job pipeline ...

Retry on failure

Repeat 3x
Test Job Actions

● A test job reaches the LAVA Dispatcher as a pipeline of actions
● The action concept within a test job definition is tightly defined
  ○ there are 3 types of actions (deploy, boot, test)
  ○ actions don’t overlap (e.g. a test action doesn’t do any booting)
  ○ Repeating an action gives the same result (idempotency)
● The pipeline structure of each job is explicit - no implied actions or behaviour
● All pipeline steps are validated at submission, this includes checks on all urls
● Actions, and the elements that make them up, are documented here
  https://validation.linaro.org/static/docs/v2/dispatcher-actions.html#dispatcher-actions
● Link to Sample Job Files
Zephyr JOB definition for NXP K64F

device_type: 'frdm-k64f'
job_name: 'zephyr tutorial 001 - from ppl.l.o'

actions:
- deploy:
  timeout:
    minutes: 3
  to: tmpfs
  images:
    zephyr:
      url: 'https://people.linaro.org/~bill.fletcher/zephyr_frdm_k64f.bin'

- boot:
  method: cmsis-dap
  timeout:
    minutes: 3

- test:
  monitors:
    - name: 'kernel_common'
      start: (tc_start\(\)|starting test)
      end: PROJECT EXECUTION
      pattern: (?P<result>(PASS|FAIL))\s-\s(?P<test_case_id>\w+).
      fixupdict:
        PASS: pass
        FAIL: fail
Getting Started with the Lab Instance

A quick tour of the web UI
https://validation.linaro.org/
“The LAVA Lab”
(Cambridge Lab - Production Instance)
Large installation with 7 workers
1.6M jobs
Reports 118 public devices
Welcome to LAVA

LAVA is an automated validation architecture primarily aimed at testing deployments of systems based around the Linux kernel on ARM devices, specifically ARMv7 and later. The current range of boards (device types) supported by this LAVA instance can be seen on the scheduler status page which includes details of how many boards of each type are 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 allow you to view current and past jobs as well as submit new jobs.
- **API** - information on how to interact with LAVA and export data from LAVA using XMLRPC.
- **Help** - documentation on using LAVA, worked examples and use cases, developing your own tests and how to administer a LAVA instance of your own.
- **Sign in** - once you are logged in, LAVA will build a profile for you which provides access to jobs you submit or mark as favourites, your bundle streams containing results of those tests and your filter or image report subscriptions which can alert you to changes in sets of results.

For access to the lab, mail a request to lava-lab-team@linaro.org
Welcome to LAVA

AVA is an automated validation architecture that helps you test your software on a wide range of boards (device types) supported by the LAVA ecosystem. We have a current range of boards available for tests and currently running jobs.

AVA components

- **Results** - viewing results of tests run
- **Scheduler** - jobs are scheduled on the LAVA servers
- **API** - information on how to interact with LAVA, either through use of the web interface or using scripting.
- **Help** - documentation on using LAVA, how to use the web interface, and how to add custom device types, and reports.
- **Profile** - you are logged in as Bill Franklin, with the ability to view or edit your report subscriptions.

Guides to LAVA

- **Introduction to LAVA**
- More about LAVA & LAVA Ecosystem

---

**WebUI - Scheduler drop-down for general status**

- **Submit Job** - can directly paste a yaml job file here
- **View All Jobs**, or jobs in various states
- **All (Active) Devices**
- **Reports** - Overall health check statistics
- **Workers** - details of Dispatcher instances
WebUI drop-down for job authentication tokens

All job submission requires authentication

- Create an authentication token: https://validation.linaro.org/api/tokens/
- Display the token hash
Getting Support/Reporting Issues

**LAVA Lab**
Tech Lead: Dave Pigott

**Support Portal**
- "Problems" -> "Report a Problem". Mention "LAVA Lab:" for correct assignment
- Tickets should prominently feature ‘LITE’ in the subject and summary
- Generally please put as much info as possible in the summary
- For e.g. VPN please include public keys with the request

**LAVA Project**
Tech Lead: Neil Williams

**Support Info**
Mailing list: lava-users@lists.linaro.org
Bugs.linaro.org (->LAVA Framework)
**lava-tool**

- the command-line tool for interacting with the various services that LAVA offers using the underlying XML-RPC mechanism
- can also be installed on any machine running a Debian-based distribution, without needing the rest of LAVA (`$ apt-get install lava-tool`)
- allows a user to interact with any LAVA instance on which they have an account
- primarily designed to assist users in manual tasks and uses keyring integration
- Basic useful lava-tool features:

  ```bash
  $ lava-tool auth-add <user@lava-server>
  $ lava-tool submit-job <user@lava-server> <job definition file>
  ```
Using lava-tool to submit a Lava Job

- This example uses a prebuilt image and job definition file
- Use a test image built for a lab-supported platform - in this case frdm-k64f - at https://people.linaro.org/~bill.fletcher/zephyr_frdm_k64f.bin
- Use the yaml job definition file here (complete version … and on next slide)
- Get an authentication token from the web UI and paste into lava-tool as prompted
  
  ```
  $ lava-tool auth-add https://first.last@validation.linaro.org
  ```

- Submit the job
  
  ```
  $ lava-tool submit-job https://first.last@validation.linaro.org zephyr_k64_job001.yaml
  ```

- lava-tool returns the job number if the submission is successful. You can follow the results at https://validation.linaro.org/scheduler/myjobs, finding the job number.
# Zephyr JOB definition for NXP K64F

device_type: 'frdm-k64f'
job_name: 'zephyr tutorial hw test job submission 001 - from ppl.l.o'

```
timeouts:
  job:
    minutes: 6
  action:
    minutes: 2
priority: medium
visibility: public

actions:
- deploy:
    timeout:
      minutes: 3
to: tmpfs
images:
  zephyr:
    url: 'https://people.linaro.org/~bill.fletcher/zephyr_frdm_k64f.bin'

- boot:
  method: cmsis-dap
  timeout:
    minutes: 3

- test:
  monitors:
    - name: 'kernel_common'
      start: (tc_start\(|\)|starting test)
      end: PROJECT EXECUTION
      pattern: (?P<result>(PASS|FAIL))\s-\s(?P<test_case_id>\w+).\s*
      fixupdict:
        PASS: pass
        FAIL: fail
```
Anatomy of a test job definition

- **Documentation**
  
  https://validation.linaro.org/static/docs/v2/explain_first_job.html

- **Example job file** k64f-kernel-common (previous slide)

- **General details** -
  - device_type - used by the Scheduler to match your job to a device
  - job_name - free text appearing in the list of jobs
  - Global timeouts - to detect and fail a hung job

- **Context:**
  - Used to set values for selected variables in the device configuration.
  - Most commonly, to tell the qemu template e.g. which architecture is being tested

- **Test Job actions:** Deploy, Boot, Test
Deploy action

- deploy:
  
  timeout:
    minutes: 3
  
  to: tmpfs
  
  images:
    
    zephyr:
      
      image_arg: '-kernel {zephyr}'
      
      url: 'https://...'

Downloads files required by the job to the dispatcher
detailed docs

  timeout: self-explanatory - can use seconds … to hours …
  
  to: specifies the deploy method
  
  image_arg: only needed for jobs that run on qemu Cortex M3
  
  url: the location of the image

Many other deploy features not used here: OS awareness, loading test overlays onto rootfs images
Boot action

- boot:
  - method: cmsis-dap
  - timeout:
    - minutes: 3

---

Boot the device

**Detailed docs**

**timeout:** self-explanatory

**method:** specifies either the command to run on the dispatcher or the interaction with the bootloader on the target

---

No Parameters?

- The individual board is not known at job submission time, so the Scheduler has to populate the relevant ports, power-reset control I/O etc
- Command line parameters for e.g. pyocd are populated from the device_type template in the Scheduler

Zephyr specific boot methods:

- **cmsis_dap.py**
- **pyocd.py**
- qemu
Test action

- test:
  - monitors:
    - name: 'kernel_common'
      start: (tc_start\(\)|starting test)
      end: PROJECT EXECUTION
      pattern: (?P<result>(PASS|FAIL))\s-\s(?P<test_case_id>\w+)
      fixupdict:
        PASS: pass
        FAIL: fail

Sample output to parse:
PASS - byteorder_test_memcpy_swap.

Execute the required tests

monitors: one-way DUT connection
- https://git.linaro.org/lava/lava-dispatcher.git/tree/lava_dispatcher/actions/test/monitor.py
  name: appears in the results output
  start: string used to detect when the test action starts
  end: string used to detect when the test action is finished
  pattern: supplies a parser that converts each test output into results
  fixupdict: as a default, LAVA understands only “pass”|”fail”|”skip”|”unknown”
Looking at LAVA results

See what happens when we run the job …

- In the following slides:
- Results
- Job Details
- Timing

$ lava-tool submit-job returns the job number...

A link to the full trace is here:
https://validation.linaro.org/scheduler/job/1656241
### Job Summary List

#### Jobs submitted by bill.fletcher

<table>
<thead>
<tr>
<th>ID</th>
<th>Actions</th>
<th>Status</th>
<th>Priority</th>
<th>Device</th>
<th>Description</th>
<th>Submitter</th>
<th>Submit time</th>
<th>End time</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1656246</td>
<td></td>
<td>Incomplete</td>
<td>Medium</td>
<td>qemu-dispatcher01</td>
<td>zephyr tutorial test job submission 003 - from ppl.l.o</td>
<td>bill.fletcher</td>
<td>Dec. 04, 3:06p.m.</td>
<td>Dec. 04, 3:06p.m.</td>
<td>0:00:01</td>
</tr>
<tr>
<td>1656245</td>
<td></td>
<td>Complete</td>
<td>Medium</td>
<td>qemu-dispatcher01</td>
<td>zephyr tutorial test job submission 003 - from ppl.l.o</td>
<td>bill.fletcher</td>
<td>Dec. 04, 3:05p.m.</td>
<td>Dec. 04, 3:06p.m.</td>
<td>0:00:03</td>
</tr>
<tr>
<td>1656241</td>
<td></td>
<td>Complete</td>
<td>Medium</td>
<td>frdm-k64f-04</td>
<td>zephyr tutorial hw test job submission 001 - from ppl.l.o</td>
<td>bill.fletcher</td>
<td>Dec. 04, 2:41p.m.</td>
<td>Dec. 04, 2:43p.m.</td>
<td>0:00:24</td>
</tr>
<tr>
<td>1656240</td>
<td></td>
<td>Complete</td>
<td>Medium</td>
<td>frdm-k64f-03</td>
<td>zephyr tutorial hw test job submission 001 - from ppl.l.o</td>
<td>bill.fletcher</td>
<td>Dec. 04, 2:41p.m.</td>
<td>Dec. 04, 2:42p.m.</td>
<td>0:00:24</td>
</tr>
<tr>
<td>1656239</td>
<td></td>
<td>Complete</td>
<td>Medium</td>
<td>frdm-k64f-04</td>
<td>zephyr tutorial hw test job submission 001 - from ppl.l.o</td>
<td>bill.fletcher</td>
<td>Dec. 04, 2:41p.m.</td>
<td>Dec. 04, 2:42p.m.</td>
<td>0:00:24</td>
</tr>
<tr>
<td>1656238</td>
<td></td>
<td>Complete</td>
<td>Medium</td>
<td>frdm-k64f-03</td>
<td>zephyr tutorial hw test job submission 001 - from ppl.l.o</td>
<td>bill.fletcher</td>
<td>Dec. 04, 2:41p.m.</td>
<td>Dec. 04, 2:42p.m.</td>
<td>0:00:24</td>
</tr>
<tr>
<td>1656237</td>
<td></td>
<td>Complete</td>
<td>Medium</td>
<td>frdm-k64f-04</td>
<td>zephyr tutorial hw test job submission 001 - from ppl.l.o</td>
<td>bill.fletcher</td>
<td>Dec. 04, 2:40p.m.</td>
<td>Dec. 04, 2:41p.m.</td>
<td>0:00:24</td>
</tr>
</tbody>
</table>
Results

LAVA Results for 1656245 Complete

Exports
- Test results: CSV or YAML
- Job metadata: CSV or YAML

Actions
- Similar jobs

Details
- Device: qemu-dispatcher01

<table>
<thead>
<tr>
<th>Job ID</th>
<th>Actions</th>
<th>Submitter</th>
<th>Test Suite</th>
<th>Passes</th>
<th>Fails</th>
<th>Totals</th>
<th>Logged</th>
<th>Bug Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>1656245</td>
<td>![Green Check]</td>
<td>bill.fletcher</td>
<td>kernel_common</td>
<td>11</td>
<td>0</td>
<td>11</td>
<td>Dec. 4, 2017, 3:06 p.m.</td>
<td>[0] (0)</td>
</tr>
<tr>
<td>1656245</td>
<td>![Green Check]</td>
<td>bill.fletcher</td>
<td>lava</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>Dec. 4, 2017, 3:06 p.m.</td>
<td>[0] (0)</td>
</tr>
</tbody>
</table>

Metadata

Notifications
Job Details - start of Deploy action

Description: zephyr tutorial hw test job submission 001 - from ppl_lo

Device-type: frdm-k64f
Submitter: Bill Fletcher
Created: 48 minutes ago
Priority: Medium
Visibility: Publicly visible

#1656241 Complete
Device: frdm-k64f-04
Started: 46 minutes ago
Duration: 0 minutes
Results:

Job Summary:
lava-dispatcher, installed at version: 2017.11.post1-1+jessie
start: 0 validate
Start time: 2017-12-04 14:42:38.255648+00:00 (UTC)
Validating that https://people.linaro.org/~bill.fletcher/zephyr_frdm_k64f.bin exists
validate duration: 0.37

case: validate
case_id: 1787969
definition: lava
result: pass

start: 1 deployimages (timeout 00:03:00) [common]
start: 1.1 download-retry (timeout 00:03:00) [common]
start: 1.1.1 http-download (timeout 00:03:00) [common]
No compression specified.
downloading https://people.linaro.org/~bill.fletcher/zephyr_frdm_k64f.bin
saving as /var/lib/lava/dispatcher/tmp/1656241/deployimages-Tcp_du/zephyr/zephyr_frdm_k64f.bin
total size: 28304 (0MB)
progress 100% (0MB)
0MB downloaded in 0.42s (0.00MB/s)
and: 1.1.1 http_download (duration 00:00:00) [common]
Job Details - start of Boot action

end: 1 deploy-images (duration 00:00:00) [common]
start: 2 boot-cmsis (timeout 00:03:00) [common]
start: 2.1 boot-cmsis-retry (timeout 00:03:00) [common]
start: 2.1.1 reset-device (timeout 00:03:00) [common]
start: 2.1.1.1 pdu-reboot (timeout 00:03:00) [common]
nice /usr/local/lab-scripts/usb_hub_control -p 7100 --mode1 off --mode2 sync -d 15 -u 08
command output Attempting to connect to device using: telnet localhost 7100
Matched 'Cambrionix USB Hub Management Interface' which means all-good
Looking for prompt
Port 08 now set to a
Telnet connection closed
Attempting to connect to device using: telnet localhost 7100
Matched 'Cambrionix USB Hub Management Interface' which means all-good
Looking for prompt
Port 98 now set to s
Telnet connection closed
end: 2.1.1.1 pdu-reboot (duration 00:00:16) [common]

case: pdu-reboot
case_id: 1787985
definition: lava
duration: 35.70
extra: ...
level: 2.1.1.1
namespace: common
result: pass

end: 2.1.1 reset-device (duration 00:00:16) [common]
start: 2.1.2 wait-device-path (timeout 00:02:44) [common]
Waiting for udev device path: /dev/disk/by-id/usb-MBED_microcontroller_024002619EA14E65035FB3DD-0:0
end: 2.1.2 wait-device-path (duration 00:00:02) [common]
start: 2.1.3 flash-cmsis (timeout 00:02:42) [common]
Job Details - Test action parsing

- test:
  - monitors:
    - name: 'kernel_common'
      start: (tc_start\(\))|starting test)
      end: PROJECT EXECUTION
      pattern:
        (?P<result>(PASS|FAIL))\s-\s(?P<test_case_id>\w+).\n        fixupdict:
        PASS: pass
        FAIL: fail
      (not matched)
Job Timing - for timeout tuning, not benchmarking

Job Timings

This section helps test writers to check for actions with a duration which is much shorter than the requested timeout. Reducing these timeouts will allow failures to be identified more quickly.

The graph only shows actions that are longer than 1 second. The full list is anyway available in the table.

**Durations**

<table>
<thead>
<tr>
<th>Action</th>
<th>Duration</th>
<th>Failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>deployimages</td>
<td>0.00%</td>
<td>0.0</td>
</tr>
<tr>
<td>boot-cmsis</td>
<td>100.00%</td>
<td>22.0</td>
</tr>
<tr>
<td>lava-test-monitor.retry</td>
<td>0.00%</td>
<td>0.0</td>
</tr>
<tr>
<td>finalize</td>
<td>0.00%</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Timings**

- Total: 22.0 seconds
- Max: 22.0 seconds
- Mean: 1.38 seconds
A More Complex Zephyr Test Example

Output the Zephyr boot time values as the result of a test, and also that the boot test succeeded (tests/benchmarks/boot_time)

tc_start() - Boot Time Measurement
Boot Result: Clock Frequency: 12 MHz
__start     : 0 cycles, 0 us
_start->main(): 5030 cycles, 419 us
_start->task : 5461 cycles, 455 us
_start->idle : 8934 cycles, 744 us
Boot Time Measurement finished
===================================================================
PASS - main.
===================================================================
PROJECT EXECUTION SUCCESSFUL

Pipeline: cascade 2 test actions
The first test action matches _start->... and picks out the microsecond values
The second test action matches PASS and picks out the test case which is main

Example Solution:
The Job Definition
The Results
The Measurements
Writing Tests

- **pattern**: expressions need to be compatible with pexpect/re (used by the Dispatcher)
- **monitor**: is for devices without a unix-style* shell. It handles output only
- **monitor**: pattern matches can populate *named Python regex groups* for test_case_id, result, measurement, units
- Obviously tests that need some interaction to boot and/or run can’t be automated with LAVA
- The **pattern**: syntax has not been designed for complex detailed parsing of test output logs. The expectation was that it would invoke (via a shell) and parse the results of scripts/commands that would do most of the heavy lifting in dealing with test suite output

*The Lava Test Shell* is used for testing devices that have a unix style shell and a writeable FS.
Writing tests - coping strategies

- Most (non-Zephyr) LAVA users craft their test invocation scripts to fit existing **pattern**: boilerplate
- Prototype **pattern**: re expressions in an offline python script before trying them in LAVA
- Debug them further in LAVA test actions on an M3 qemu instance first (fast, doesn’t tie up resources, unbreakable)
- The more carefully crafted a **pattern**: is, the more brittle it will likely be when the Zephyr-side code changes
- Cascading multiple test action blocks can solve more complex parsing problems
LAVA and CI

Overview
LAVA in ci.linaro.org
XMLRPC
Metadata
Job templates
Overview - industrializing LAVA

Health checks
Target requirements
Metadata
Health Checks & Gold Standard Images

- **Health check**
  - special type of test job
  - designed to validate a test device and the infrastructure around it
  - run periodically to check for equipment and/or infrastructure failures
  - needs to at least check that the device will boot and deploy a test image.

- **Writing Health Checks**
  - It has a job name describing the test as a health check
  - It has a minimal set of test definitions
  - It uses [gold standard files](#)

- **Gold Standard**
  - Gold standard has been defined in association with the QA team.
  - Provide a known baseline for test definition writers
  - (open point: are there gold standard images and jobs for LITE target boards?)
Sources of Target Board Success ...

- See https://validation.linaro.org/static/docs/v2/device-integration.html section on Device Integration

A few LITE-relevant points:

- **Serial**
  - Persistent, stable
  - if over a shared OTG cable, other traffic does not disrupt trace

- **Reset**
  - Image data not retained
  - ‘old’ serial data not buffered/retained

- **Predictable & repeatable**

- **No manual intervention**
**Metadata**

- Linking a LAVA job and its result artifacts back to the code - not important for ad hoc submission, but vital for CI
- Specific **metadata**: section within the jobfile
- Can be queried for a job via xmlrpc
- Example API call: `get_testjob_metadata(job_id)`
- Call returns entries created by LAVA as well as submitted in the test job definition
- Example

```plaintext
metadata:
  build-url: $build_url
  build-log: $build_url/consoleText
  zephyr-gcc-variant: $gcc_variant
  platform: $board_name
  git-url: https://git.linaro.org/zephyrproject-org/zephyr.git
  git-commit: $git_commit
```
**LAVA in ci.linaro.org**

Idealised flow:

- In practice, LAVA jobs are submitted by the QA server which acts as a proxy, not by ci.linaro.org

- In either case LAVA is invoked via xmlrpc APIs
Invoking a LAVA job via xmlrpc

#!/usr/lib/python

import xmlrpclib

username = "bill.fletcher"
token = "<token string>"
hostname = "validation.linaro.org"
server = xmlrpclib.ServerProxy("https://%s:%s@%s/RPC2" % (username, token, hostname))

jobfile = open("zephyr_k64_job001.yaml")
jobtext = jobfile.read()

id = server.scheduler.submit_job(jobtext)
print server.scheduler.job_status(id)

The above is approximately equivalent to $ lava-tool submit-job ...
The API is documented here https://validation.linaro.org/api/help/
Creating the jobfile on the fly - templates

Uses `class string.Template(template)`

template_file_name = "lava-job-definitions/%s/template.yaml" % (args.device_type, )
    test_template = None
if os.path.exists(template_file_name):
    test_template_file = open(template_file_name, "r")
    test_template = test_template_file.read()
    test_template_file.close()
else:
    sys.exit(1)

replace_dict = dict(
    build_url=args.build_url,
    test_url=args.test_url,
    device_type=args.device_type,
    board_name=args.board_name,
    test_name=args.test_name,
    git_commit=args.git_commit,
    gcc_variant=args.gcc_variant
)

_template = Template(test_template)
lava_job = template.substitute(replace_dict)
Job Templates - actions

actions:
- deploy:
  timeout:
   minutes: 3
  to: tmpfs
  images:
    zephyr:
      url: '$test_url'

- boot:
  method: pyocd
  timeout:
    minutes: 10

- test:
  timeout:
    minutes: 10
  monitors:
  - name: '$test_name'
    start: (tc_start\(|\w*starting .*test|BOOTING ZEPHYR OS)
    end: PROJECT EXECUTION
    pattern: (\w*PASS|\w*FAIL)\s*\(\s*\w*\w*\)\/\.
    fixupdict:
      PASS: pass
      FAIL: fail

Maybe consider also including pattern: in the template, so that it tracks any changes in the test
# Zephyr JOB definition for frdm-kw41z

```plaintext
device_type: ' ${device_type}'
job_name: 'zephyr-upstream $test_name'

timeouts:
  job:
    minutes: 30
  action:
    minutes: 3
  actions:
    wait-usb-device:
      seconds: 40

priority: medium
visibility: public

<actions>

metadata:
  build-url:  $build_url
  build-log:  $build_url/consoleText
  zephyr-gcc-variant:  $gcc_variant
  platform:  $board_name
  git-url:  https://git.linaro.org/zephyrproject-org/zephyr.git
  git-commit:  $git_commit
```
Thank You

#HKG18

HKG18 keynotes and videos on: connect.linaro.org
For further information: www.linaro.org