



Getting Started in LAVA

Linaro

Automated

Validation

Architecture



How is Linaro Using LAVA?

- Daily automated tests of multiple images and hardware packs on Linaro member hardware
- Continuous Integration testing on Linaro Android images
- Continuous Integration testing of multiple kernel trees/config/boards



Overview: Major LAVA Components

- LAVA Server
- LAVA Dashboard
- LAVA Scheduler
- LAVA Dispatcher
- LAVA Test
- ...and many other tools and libraries that I'll completely gloss over here in the interest of time



LAVA Scheduler

- Primary entry point for test jobs
- Displays current device status and current job status
- Provides XML-RPC API for job submission
- Provides command-line tools for job submission
- Live monitoring of jobs in progress
- Admin functions for taking down boards for maintenance and canceling jobs

LAVA Server Dashboard Scheduler API
You are here: LAVA > Scheduler

Devices

Show 10 entries Search:

Type	Hostname	Status
beaglexm	beaglexm02	Idle
beaglexm	beaglexm01	Idle
panda	panda01	Running
panda	panda02	Running

Showing 1 to 4 of 4 entries

Active Jobs

Show 10 entries Search:

ID	Status	Target	Submitter	Submit Time
52	Running	plalco		July 27, 2011, 4:43 p.m.
53	Running	plars		July 27, 2011, 5:23 p.m.

Showing 1 to 2 of 2 entries

LAVA Dashboard

- Storage and retrieval of raw test results and attachments
- Provides XML-RPC API for results submission
- Provides command-line interface to the API

Some Terminology:

- Bundle Stream - A way of organizing related result bundles
- Result Bundle - a set of results submitted after a testing session, can contain multiple test runs, as well as other information about the system where the testing was performed
- Test Run - The results from a single test suite
- Test Case - The individual id and result of a single test within a test run



LAVA Server

- Server framework for LAVA web components
- Extensions can be written to add functionality
- Dashboard and Scheduler are the primary extensions
- Extensions can also be written to do things like special data or results handling (example: Kernel CI results)



LAVA Dispatcher

- Talks to the individual test devices
- Executes actions such as deploying images, running tests, and submitting results
- Currently supports Linaro and Android image deployments
- Local board configuration details are stored in config files
- Can be used standalone, or kicked off by the scheduler daemon



LAVA Test

- Provides a uniform interface for installing tests, running them, and parsing results
- Test wrappers act as the "glue" between individual test suites and LAVA
- Supports out-of-tree tests for tests that can't be included
- Test parameter defaults can be overridden
- Can be used standalone for convenient local execution

LAVA Android Test

- Provides the Above functionality for Android testing
- Uses ADB to talk to Android devices rather than running locally



Running a test

- Install lava-test from pypi, bzt, or package (linaro-validation ppa)

```
$ pip install --user lava-test
```

```
$ lava-test install stream
```

```
$ lava-test run stream
```



How can I add my test?

- LAVA Test (or LAVA Android Test)
- Test wrapper simply defines the basics of your test
- Let's see an example:

```
from lava_test.core.installers import TestInstaller
from lava_test.core.parsers import TestParser
from lava_test.core.runners import TestRunner
from lava_test.core.tests import Test
```

```
URL="http://www.cs.virginia.edu/stream/FTP/Code/stream.c"
INSTALLSTEPS = ['cc stream.c -O2 -fopenmp -o stream']
DEPS = ['gcc', 'build-essential']
DEFAULT_OPTIONS = ""
RUNSTEPS = ['./stream ${OPTIONS}']
PATTERN = "^(?P<test_case_id>\w+):\W+(?P<measurement>\d+\.\d+)"
```

```
streaminst = TestInstaller(INSTALLSTEPS, deps=DEPS, url=URL)
streamrun = TestRunner(RUNSTEPS, default_options=DEFAULT_OPTIONS)
streamparser = TestParser(PATTERN, appendall={'units':'MB/s', 'result':'pass'})
testobj = Test(test_id="stream", installer=streaminst, runner=streamrun, parser=streamparser)
```



Pattern fields

- Patterns are defined using regular expressions
- Fields matched correspond to result bundle fields

```
PATTERN = "((?P<test_case_id>\A(\w+[/]+)\w+[-]*\w*[-]*\w*) .*? (?P<result>\w+))"
```

```
PATTERN = "^(?P<test_case_id>\w+):\W+(?P<measurement>\d+\.\d+)"
```

Mandatory fields:

- test_case_id (text field)
- result (see previous slide for values)

Other useful fields:

- measurement - for benchmarks
- units - used for measurements (ex. MB/s, foos/bar)



Adding things to every result

- Sometimes results have a value you just "know" but can't be parsed
- Sometimes all results have something in common

```
my_parser = TestParser(PATTERN, appendall={'units':'ms','result':'pass'})
```



Tips for test_case_id fields

- You can have 100,000 identical test_case_id's with their own results in the same test run (but this probably isn't what you want)
- IDs help someone looking at the results to identify which testcase this result is describing (this may include sub-ids, or parameters)
- For tests that do the same thing every time, and vary only on parameters, the id might encode the parameters

Overall, you want to ensure that when looking at two different test runs, the **SAME** test_case_id is used to describe the same test case being run (otherwise it will be very confusing if you ever want to graph, compare, etc)



An example: LTP

- LTP Tests commonly have test cases, with sub test cases within them
- Without dealing with this, you might end up with multiple results for the same test_case_id
- Solution: encode sub-test-id in the test_case_id
- Example:

fttruncate03.1

fttruncate03.2

fttruncate03.3



Adapting Test Results

- LAVA supported test results are: pass, fail, skip, unknown
- example, posixtestsuite contains very different results:

```
PATTERN = "((?P<test_case_id>\A(\w+[/]+)\w+[-]*\w*[-]*\w*) .*? (?P<result>\w+))"
```

```
FIXUPS = {  
    "FAILED"      : "fail",  
    "INTERRUPTED" : "skip",  
    "PASSED"      : "pass",  
    "UNRESOLVED"  : "unknown",  
    "UNSUPPORTED" : "skip",  
    "UNTESTED"    : "skip",  
    "SKIPPING"    : "skip"  
}
```

```
posixparser = PosixParser(PATTERN, fixupdict = FIXUPS)
```



Strategies for Complex Test Results

- Some test results can't be easily parsed with a run line rule
 - Results that span multiple lines
 - Results that must be accumulated over multiple passes
 - Results that must be heavily modified
 - Results that contain lines that would match your pattern, but aren't actual result lines
- One option, is to inherit TestParser and bend it to your will
 - Will require a bit of extra python in your wrapper
- Another option: run your test through a filter, or script it to output things in a nicer way that you *can* easily deal with



What if my test can't be public?

- Out of tree tests can be used for handling private tests, or tests with licensing restrictions
- Take care that you don't publish results to a public location (yes, dashboard can store private bundle streams)
- Option 1: Create a normal python test wrapper, with a setup file that specifies: entry points for `lava_test.test_definitions`
 - ex: see <https://code.launchpad.net/linaro-graphics-tests>
 - This can be installed from bzd, package, pypi, whatever and just used by `lava-test`
- Option 2: for *very* simple tests, json based out-of-tree test definitions.
 - These can be registered via a URL, then used normally by `lava-test`
- **RECOMMENDATION:** If you are writing an out of tree test, use a name that will keep it from getting confused (ex: `graphicswg.x11perf`)



Example JSON Test Wrapper

```
{
  "format": "LAVA-Test Test Definition Format",
  "test_id": "stream-json",
  "install": {
    "url": "http://www.cs.virginia.edu/stream/FTP/Code/stream.c",
    "steps": ["cc stream.c -O2 -fopenmp -o stream"]
  },
  "run": {
    "steps": ["/./stream"]
  },
  "parse": {
    "pattern": "^(?P<test_case_id>\\w+):\\W+(?P<measurement>\\d+\\.\\d+)",
    "appendall": {
      "units": "MB/s",
      "result": "pass"
    }
  }
}
```



How do I get these running in the lab?

- First, get your test into lava test or an out-of-tree wrapper
- Next, decide when/where/how often your test should run
 - Daily image testing
 - Android CI testing
 - Kernel CI testing
 - Your own periodic runs
- Talk to the Validation team, we'll be happy to help make sure your tests get in the right place



Visualization

- Several options exist for visualizing test results
- Basic table view with column sorting and filtering - easy, and available with no additional work
- Simple charts, tables adapted to your results
 - data view - parameterized db query defined in xml
 - data report - small javascript snippet to display results of query in the data view
 - These can be integrated into lava web server pages
 - See examples in the "Reports" section of the dashboard
- Custom LAVA Extension
 - More advanced reporting
 - Can include multiple pages, multiple views, controls that modify the views
 - Can include additional data models
 - Example: Kernel CI Results extension



Linaro

CONNECT

