CE Workgroup

Fuego: Introduction, Status and Future Directions

Tim Bird

Architecture Group Chair

LF CE Workgroup

Two presentations in one

- Introduction to Fuego
  - For people learning Fuego
- Status and Future Directions
  - For people interested in open source test frameworks

CE Workgroup

Introduction to the Fuego Test System

Tim Bird

Architecture Group Chair

LF Core Embedded Linux Project

Outline

Introduction

Architecture

Customization

Vision

Introduction

Fuego = Jenkins +

Introduction

Fuego = Jenkins + abstraction scripts +

Introduction

Fuego = Jenkins + abstraction scripts + pre-packaged tests

Introduction

Fuego = (Jenkins + abstraction scripts + pre-packaged tests) inside a container

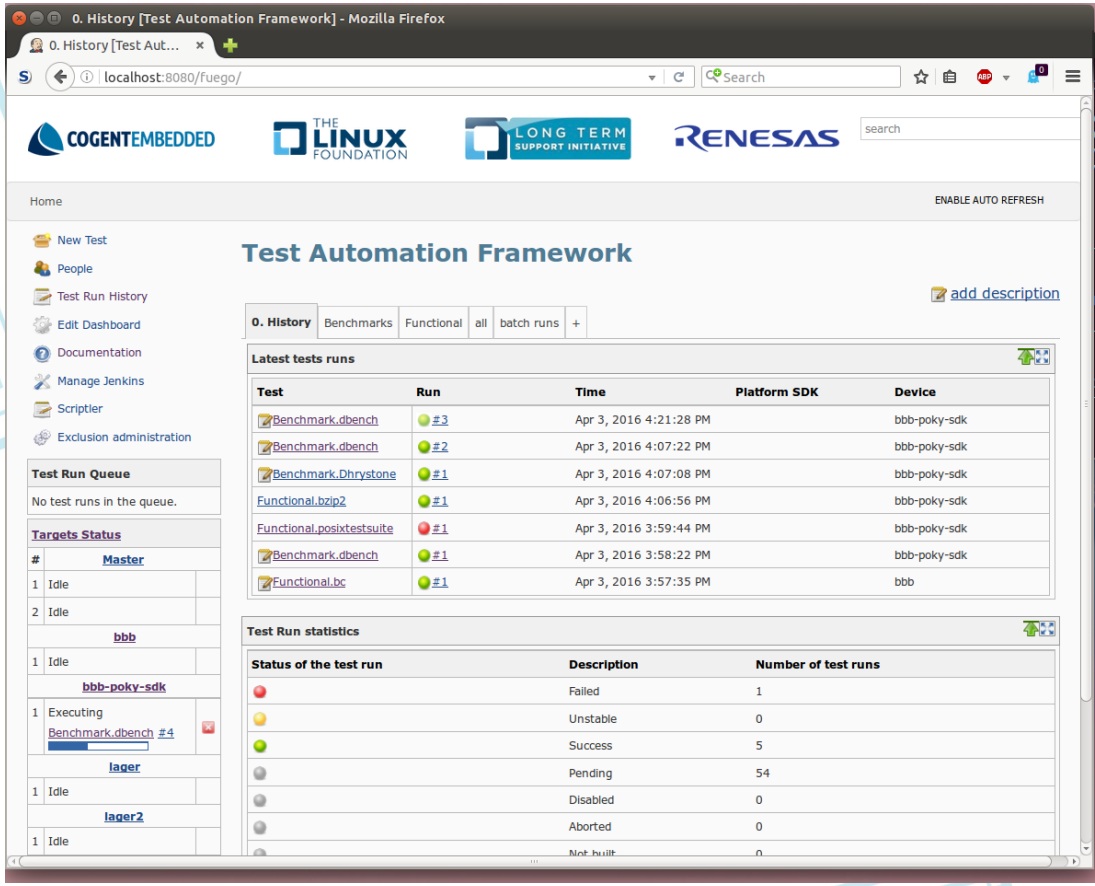
Jenkins

- Is a Continuous Integration system
  - Launches test jobs based on various triggers
  - Shows test results
  - Has an ecosystem of plugins for extended functionality
    - Integration with different source code management systems
    - E-mail notifications
    - Different interface views
    - Plotting of results
- Is too big a system to describe in detail here

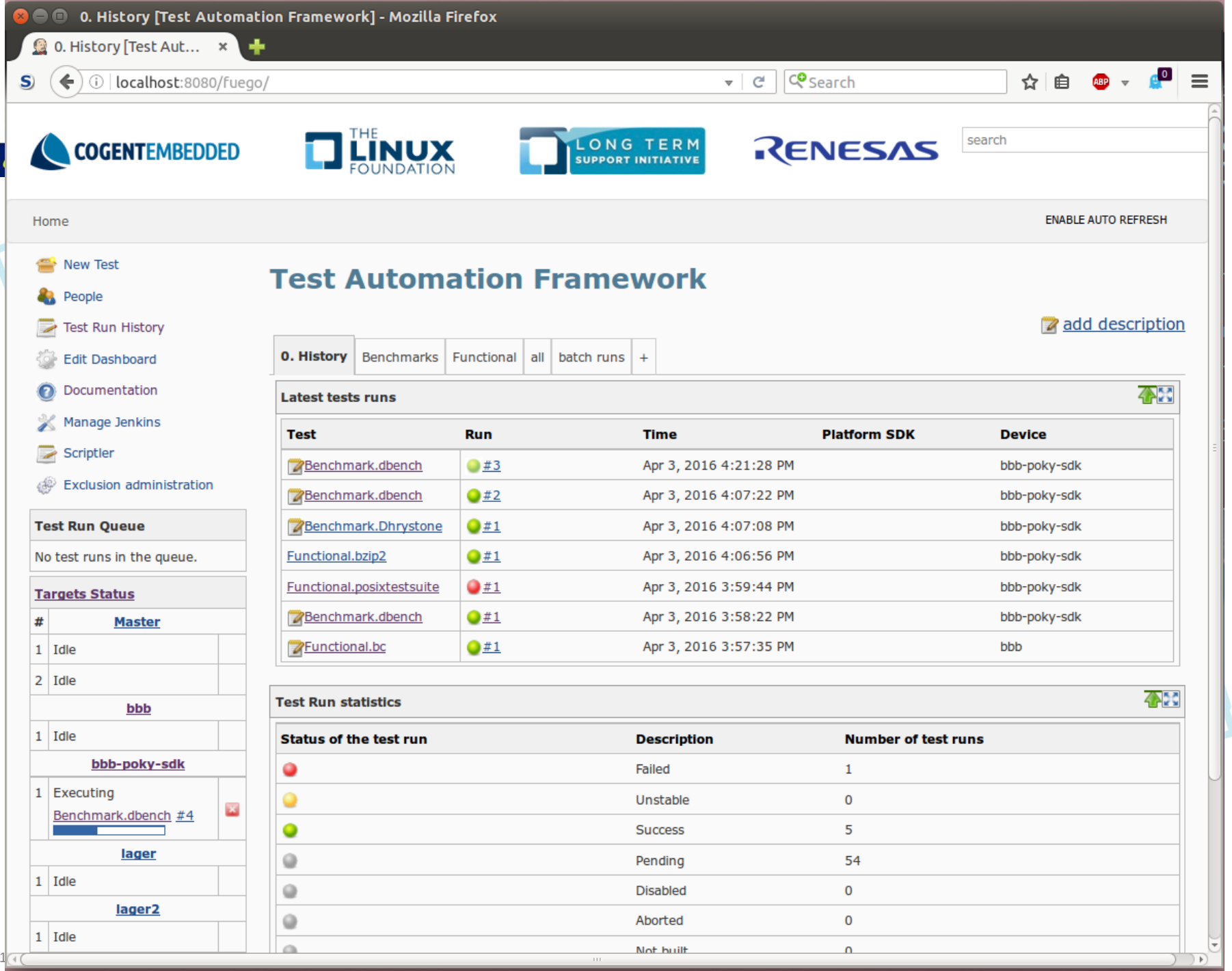
Jenkins

- Base interface:

Test history and test selection dashboard



- Fuego includes customizations to Jenkins to support host/target test configurations
- Pre-install plugins for interface changes, plotting and other stuff


0. History [Test Automation Framework] - Mozilla Firefox
0. History [Test Aut...
localhost:8080/fuego/
Search
COGENTEMBEDDED
THE LINUX FOUNDATION
LONG TERM SUPPORT INITIATIVE
RENESAS
search
Home
ENABLE AUTO REFRESH
New Test
People
Test Run History
Edit Dashboard
Documentation
Manage Jenkins
Scriptler
Exclusion administration
Test Run Queue
No test runs in the queue.
Targets Status
Master
1 Idle
2 Idle
bbb
1 Idle
bbb-poky-sdk
1 Executing
Benchmark.dbench #4
lager
1 Idle
lager2
1 Idle
Test Automation Framework
add description
0. History
Benchmarks
Functional
all
batch runs
+
Latest tests runs
Test
Run
Time
Platform SDK
Device
Benchmark.dbench
#3
Apr 3, 2016 4:21:28 PM
bbb-poky-sdk
Benchmark.dbench
#2
Apr 3, 2016 4:07:22 PM
bbb-poky-sdk
Benchmark.Dhrystone
#1
Apr 3, 2016 4:07:08 PM
bbb-poky-sdk
Functional.bzip2
#1
Apr 3, 2016 4:06:56 PM
bbb-poky-sdk
Functional.posixtestsuite
#1
Apr 3, 2016 3:59:44 PM
bbb-poky-sdk
Benchmark.dbench
#1
Apr 3, 2016 3:58:22 PM
bbb-poky-sdk
Functional.bc
#1
Apr 3, 2016 3:57:35 PM
bbb
Test Run statistics
Status of the test run
Description
Number of test runs
Failed
1
Unstable
0
Success
5
Pending
54
Disabled
0
Aborted
0
Not built
0

Abstraction scripts

- User defines a few variables in shell scripts, to allow system to interact with target boards
- Fuego provides shell functions for command and control of target:
  - Put/get files, execute commands, collect logs, etc.
- Fuego generates a full test script at runtime, based on board configuration, toolchain variables, and test variables
  - This allows all aspects of tests to be abstracted
    - This is a bigger deal than it sounds like

Pre-packaged tests

- Comes with over 50 tests, already integrated
  - aim7, blobsalad, bonnie, cyclitest, dbench, dhrystone, ebizzy, ffsb, fio, GLMark, gtkperf, hackbench, himeno, Interbench, IOzone, iperf, Java, linpack, lmbench2, nbench, netperf, netpipe, OpenSSL, reboot, signaltest, Stream, tiobench, whetstone, x11perf, aiostress, arch_timer, bzip2, cmt, crashme, expat, fontconfig, glib, ipv6connect, jpeg, libpng, linus_stress, LTP, netperf, posixtestsuite, rmaptest, scifab, scrashme, sdhi_o, stress, synctest, zlib
- Includes functional, benchmark and stress tests

Test building

- Tests are built from source
- You can use your own toolchain (/sdk)
  - Or use a pre-installed generic arm toolchain
- There's an Open Embedded meta-layer available, to help you build your own SDK in Yocto Project/Open Embedded
  - Generated SDK will have libraries and headers needed for building all tests

Inside a container

- Fuego builds a docker container
- This avoids a lot of install issues
  - Fuego can run on any Linux distro
- Builds of the test programs are 100% reproducible

Outline

Introduction

Architecture

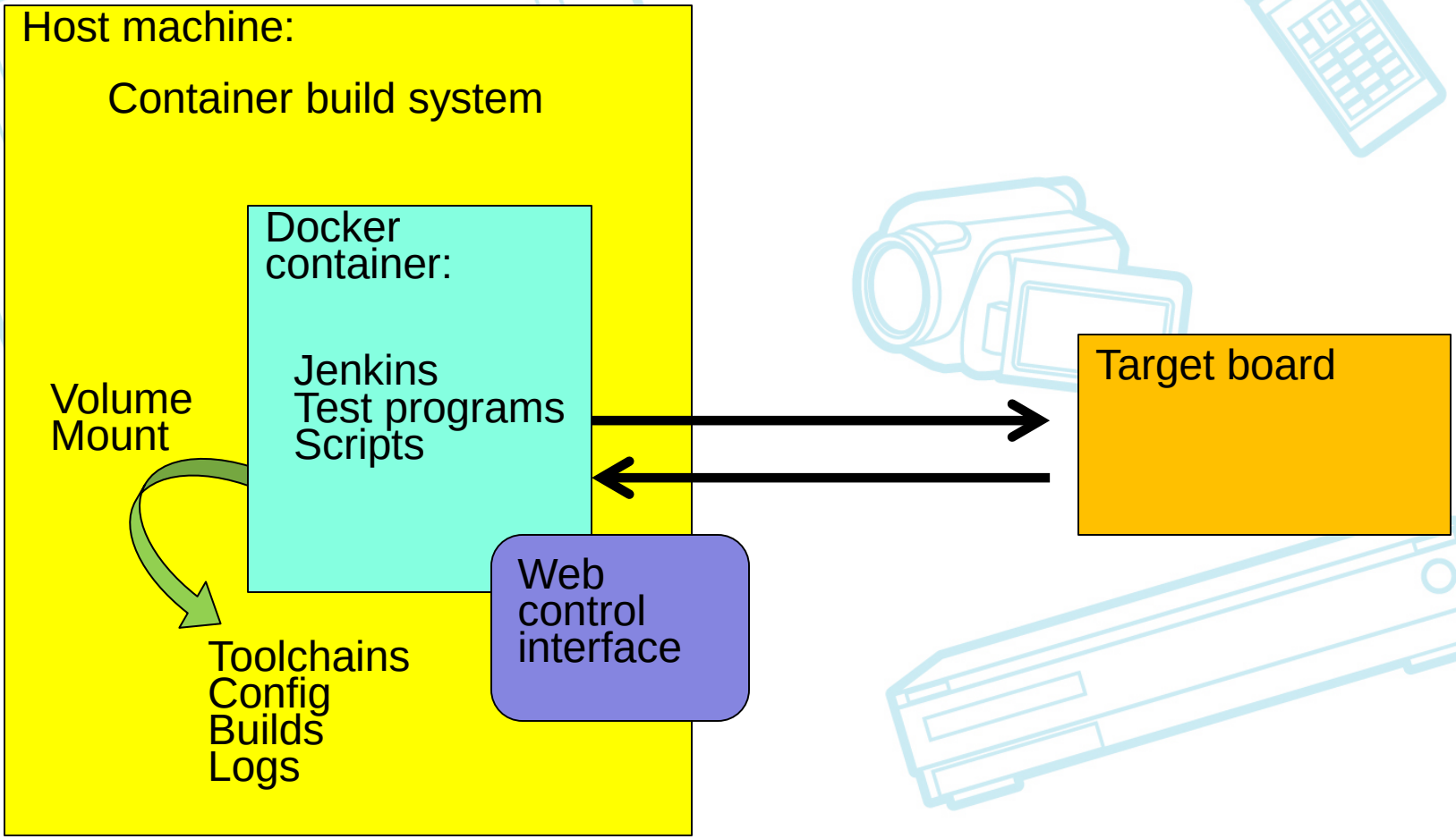
Customization

Vision

Architecture

- 2 major parts used for configuration:
  - Jenkins front-end
  - Script back-end
- Back-end is (mostly) shell-script based
  - Main interface between Jenkins and test programs is a single shell script
  - Shell is lowest common denominator language
- Very small files (glue layer) required for:
  - Log parsing
  - Results plotting

Architecture Diagram



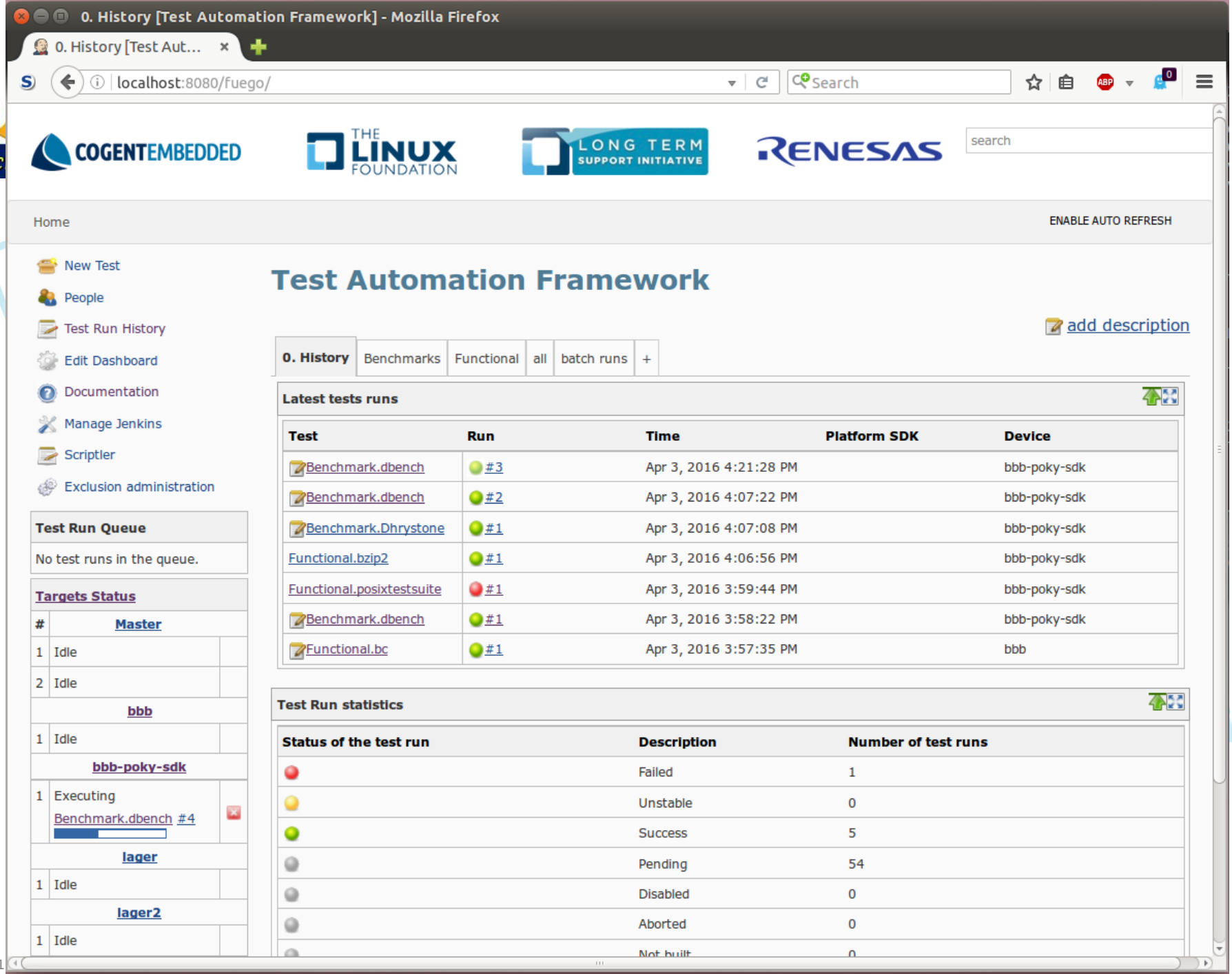
How deployed

- Comes as 2 git repositories:
  - ‘fuego’ repository - Stuff outside the container
    - Container build system
      - Including some Jenkins plugins
    - Default config and boards
    - Host scripts for controlling the container
    - Documentation
  - ‘fuego-core’ repository - Stuff inside the container
    - Script and overlay engine
    - Pre-packaged tests
    - More jenkins extensions
- Fuego-core is downloaded for you during the container image build

Getting it and using it

- git clone https://bitbucket.org/tbird20d/fuego.git
- cd fuego ; ./install.sh
  (wait a bit)
- fuego-host-scripts/docker-create-container.sh
- fuego-host-scripts/docker-start-container.sh
- firefox http://localhost:8080/fuego

- Optionally, to get additional shell prompts inside the container:
  - docker exec -i –t <*container_id*> bash
  - sshd <*user*>@localhost –p 2222
    - Requires that you create a user account inside the container


0. History [Test Automation Framework] - Mozilla Firefox
0. History [Test Aut...
localhost:8080/fuego/
Search
COGENTEMBEDDED
THE LINUX FOUNDATION
LONG TERM SUPPORT INITIATIVE
RENESAS
search
Home
ENABLE AUTO REFRESH
New Test
People
Test Run History
Edit Dashboard
Documentation
Manage Jenkins
Scriptler
Exclusion administration
Test Run Queue
No test runs in the queue.
Targets Status
Master
1 Idle
2 Idle
bbb
1 Idle
bbb-poky-sdk
1 Executing
Benchmark.dbench #4
lager
1 Idle
lager2
1 Idle
Test Automation Framework
add description
0. History
Benchmarks
Functional
all
batch runs
+
Latest tests runs
Test
Run
Time
Platform SDK
Device
Benchmark.dbench
#3
Apr 3, 2016 4:21:28 PM
bbb-poky-sdk
Benchmark.dbench
#2
Apr 3, 2016 4:07:22 PM
bbb-poky-sdk
Benchmark.Dhrystone
#1
Apr 3, 2016 4:07:08 PM
bbb-poky-sdk
Functional.bzip2
#1
Apr 3, 2016 4:06:56 PM
bbb-poky-sdk
Functional.posixtestsuite
#1
Apr 3, 2016 3:59:44 PM
bbb-poky-sdk
Benchmark.dbench
#1
Apr 3, 2016 3:58:22 PM
bbb-poky-sdk
Functional.bc
#1
Apr 3, 2016 3:57:35 PM
bbb
Test Run statistics
Status of the test run
Description
Number of test runs
Failed
1
Unstable
0
Success
5
Pending
54
Disabled
0
Aborted
0

Architecture details

- How a test is defined
- Test phases
- Overlay generation
- Test parameter abstraction

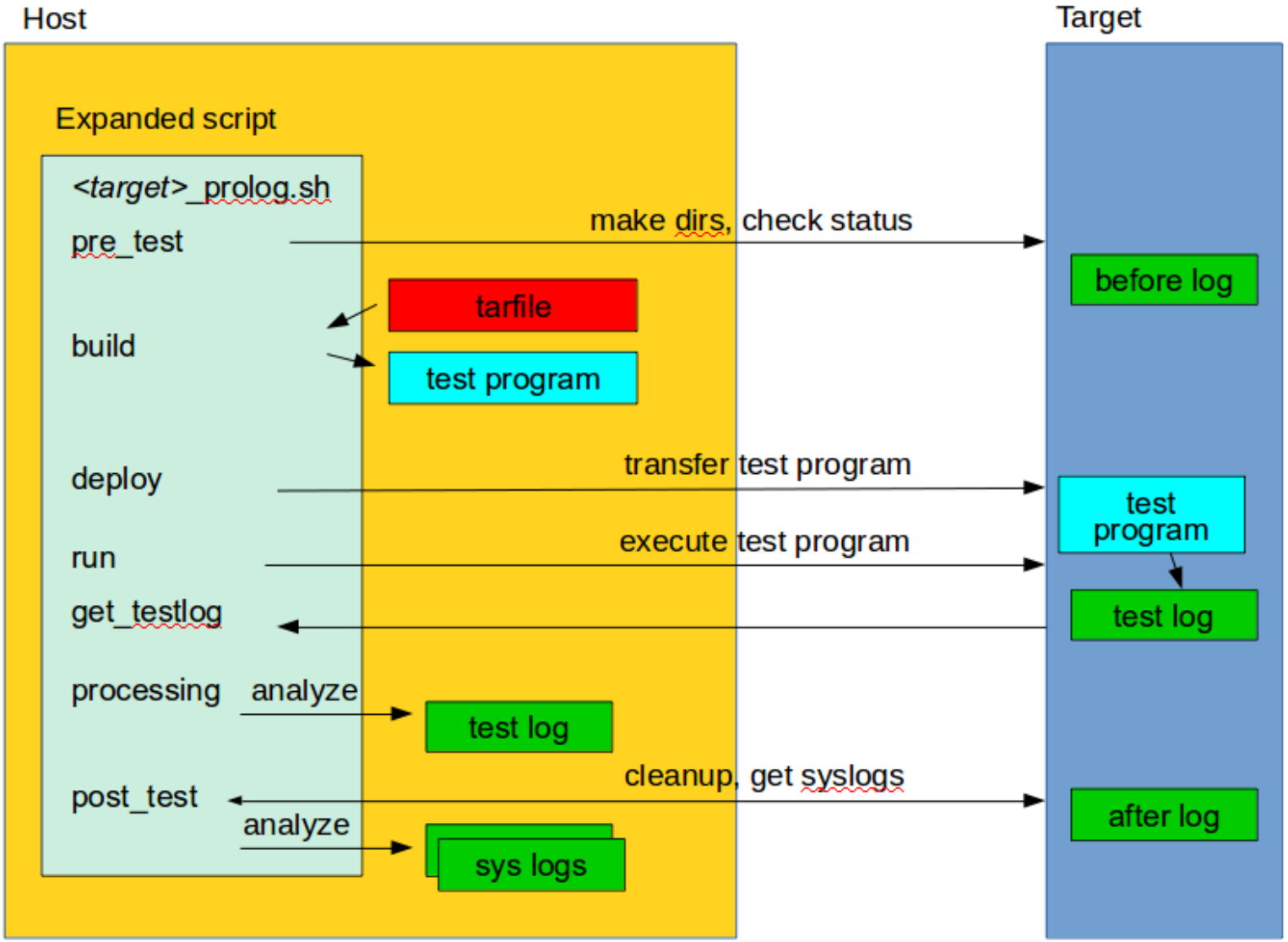
Test definition

- A Fuego test consists of:
  - Jenkins test definition – defines variables needed by Jenkins to execute the test
  - Base script – a shell script which runs on the host, which controls the execution of the test
  - Test program - an executable or script to run on the target
  - Test variables – test specs and test plans that are used to control the test
  - Results parser – tells the system how to interpret results from the test log

Test Phases

- Each test executes through phases
  - Pre_test – prepare target, check for dependencies
  - Build – compile the test program
  - Deploy – transfer the test program (and associated materials) to the target
  - Run – execute the program, on target, logging the results
  - Processing – collect the logs and parse for results
  - Post_test – clean up target and finalize Jenkins job status
- Phase can be empty if not needed:
  - For example, if no build step is needed

Fuego test phases



Test execution flow

1. Jenkins initiates a test (Jenkins test job)
   - Based on user input or some condition or trigger
2. Jenkins job starts the base script for a test
   - Overlay generator creates an expanded script and sources that into the base script
   - The test script executes through the test phases
3. Jenkins collects the console log during execution, and times the result
   - Fuego scripts collect the test log and parse the results
4. Jenkins executes the post_test step, using the expanded script
   - More logs are collected
   - Jenkins job status is updated
5. Jenkins interface can be used to see test results

Shell script example

```
#!/bin/bash

tarball=hello-test-1.1.tgz

function test_build {
        make && touch test_suite_ready || build_error "error while building test"
}

function test_deploy {
        put hello  $FUEGO_HOME/fuego.$TESTDIR/
}

function test_run {
        report "cd $FUEGO_HOME/fuego.$TESTDIR; ./hello $FUNCTIONAL_HELLO_WORLD_ARG“
}

function test_processing {
    log_compare "$TESTDIR" "1" "SUCCESS" "p"
}

. $FUEGO_SCRIPTS_PATH/functional.sh
```

Overlay generation

- Each test has a simple base script
- Fuego generates the test environment (expanded script) at test execution time using an overlay generator
  - Kind of like “object oriented” programming for shell scripts
- Four areas of overlayed functions and variables
  - Functions to interact with target
    - Board definitions
  - Toolchain variables
  - Test parameters
- Indirection for test program parameters

Overlay processing

Base script

test-script.sh

test_build()
test_deploy()
test_run()
test_processing()

Fuego functions

functional.sh

functions.sh
common.sh
overlays.sh
reports.sh
etc.

<*board*>.conf

tools.sh

testplan

test specs

ovgen.py

Expanded script

<*target*>_prolog.sh

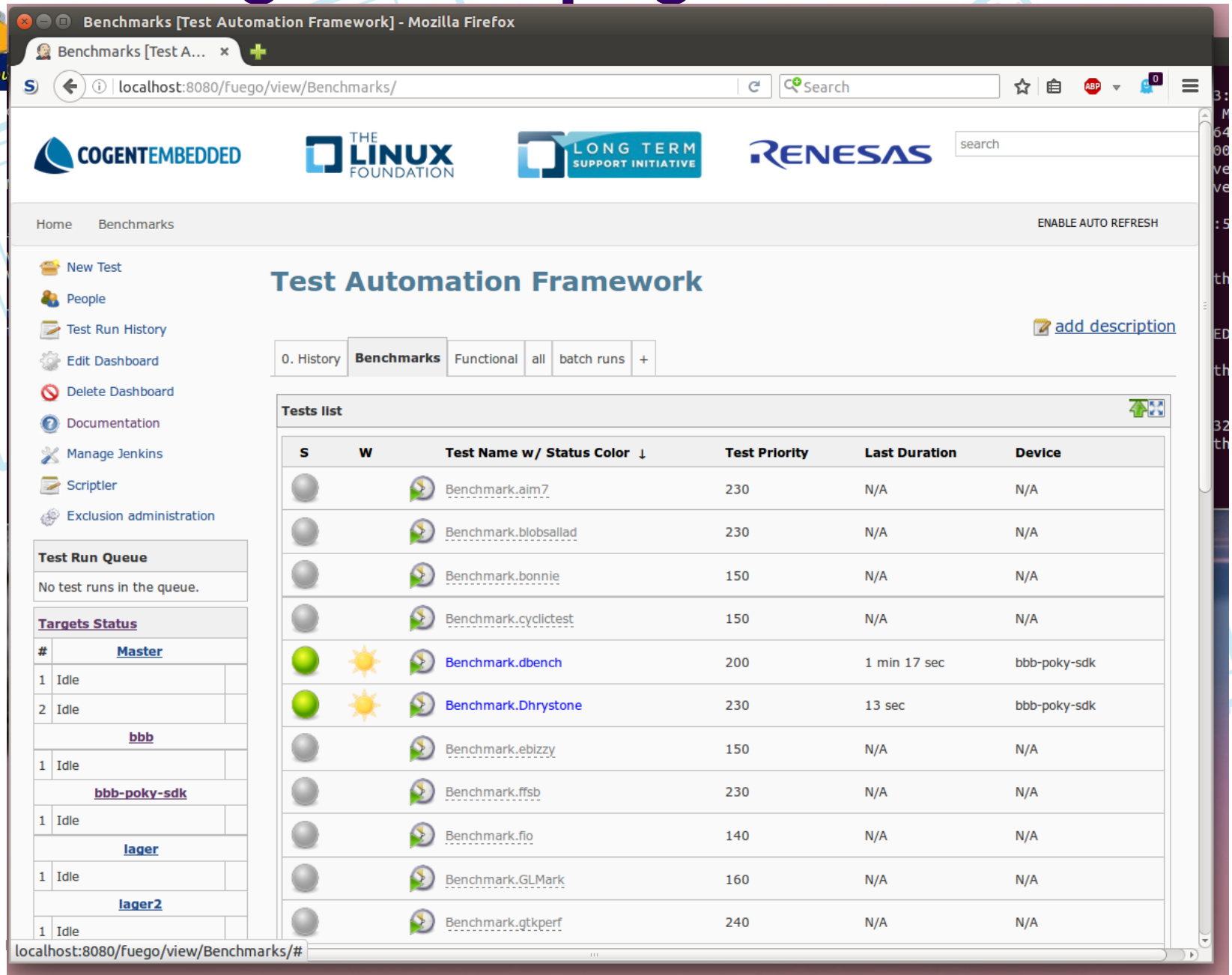
Test parameter abstraction

- Abstractions mean tests can run in multiple configurations
- Fuego abstracts details about the target board and toolchain:
  - IP address, login, target access methods
  - PLATFORM indicates toolchain to use
- Fuego also abstracts:
  - Filesystem device and mount points
  - Test program arguments
  - Expected results
- User can add new items to be abstracted, through test spec/test plan system

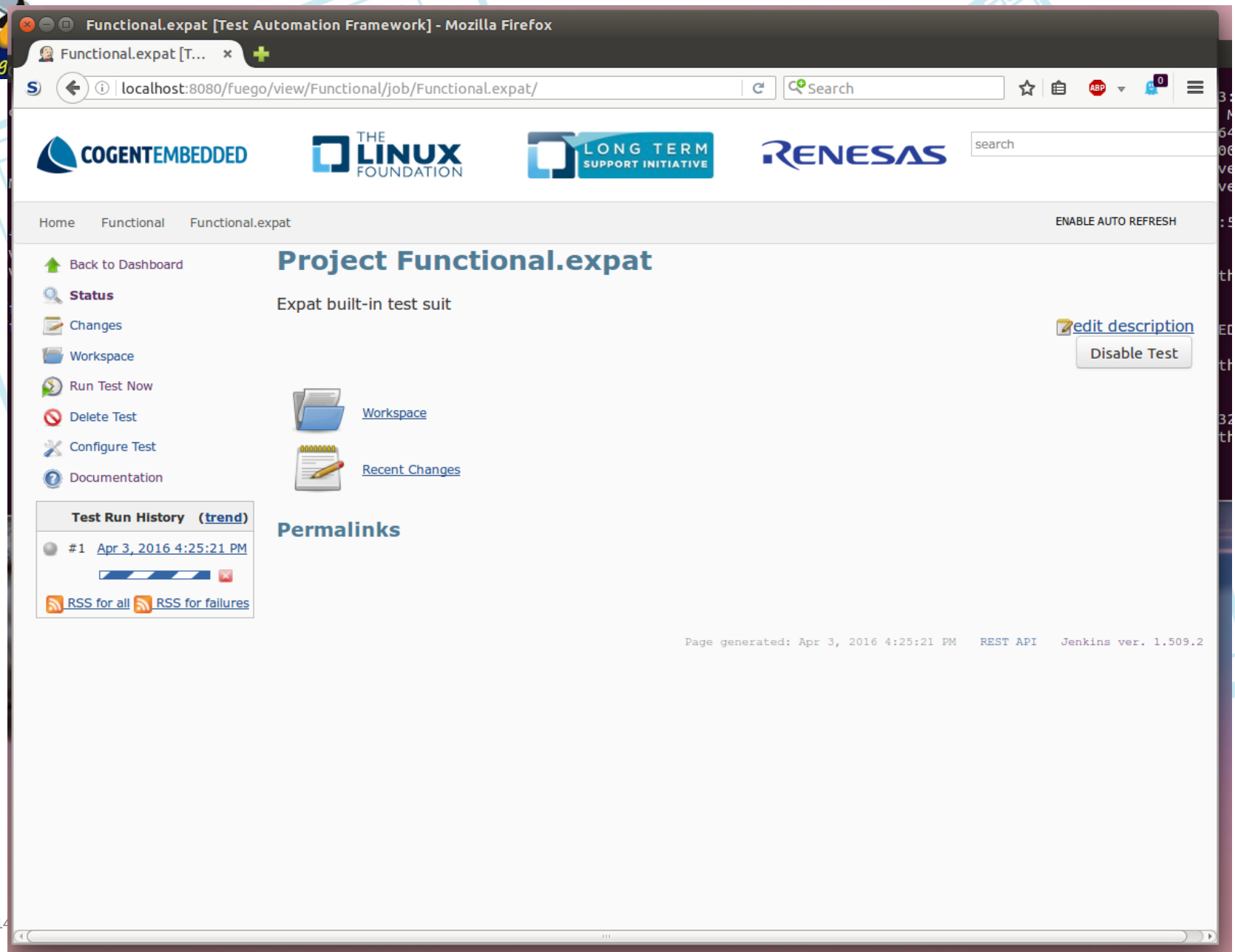
Running a test (manually)

- Select a test
- Select the target
- Select the testplan
- Push “Run the test”

Fuego tests page



Individual test page



Outline

Introduction
Architecture
Customization
Vision

Customization

- Add a board configuration
- Add a toolchain
- Add a test

Add a board

- Overview:
  - Add a board file
  - Add the new target in the Jenkins interface

The board file

- Board file is a shell script with some variable that describe the board
- Create file in userdata/conf/boards, with filename “<target-name>.board”
  - There are examples there already
- Define IP address, ssh port, file system info (device, partitions, etc.)
- PLATFORM - indicates which SDK to use for building test programs

Board file sample (qemu-arm)

```
inherit "base-board"
include "base-params"

IPADDR="172.17.0.1"
SSH_PORT=5555
LOGIN="root"
FUEGO_HOME="/home/a"
PASSWORD="adm"
PLATFORM="qemu-armv7hf"
TRANSPORT="ssh"
ARCHITECTURE="arm"

SATA_DEV="/dev/sdb1"
SATA_MP="/mnt/sata"

USB_DEV="/dev/sda1"
USB_MP="/mnt/usb"

MMC_DEV="/dev/mmcblk0p2"
MMC_MP="/mnt/mmc"

LTP_OPEN_POSIX_SUBTEST_COUNT_POS="1319"
LTP_OPEN_POSIX_SUBTEST_COUNT_NEG="169"

EXPAT_SUBTEST_COUNT_POS="1769“”
EXPAT_SUBTEST_COUNT_NEG="41"
```

Add the target in Jenkins

- Go to Target Status in main screen
- Select “New Node”
  - Enter name, and copy from “template-dev”
- Reference the board file
  - Set Environment Variable BOARD_OVERLAY to “boards/<target-name>.board”

Interface for adding a board

raspberry-pi Configuration [Test Automation Framework] - Mozilla Firefox

raspberry-pi Config...

localhost:8080/fuego/computer/raspberry-pi/configure

Search

COGENTEMBEDDED THE LINUX FOUNDATION LONG TERM SUPPORT INITIATIVE RENESAS search

Home nodes raspberry-pi

- Back to List
- Target Status
- Delete Target
- **Configure Target**
- Test Run History
- Load Statistics
- Script Console
- Log
- System Information

Targets Status

#	Status

Name: blueberry-pi

Description:

# of executors: 1

Remote FS root: /tmp/dev-slave1

Labels:

Usage: Utilize this target as much as possible

Launch method: Launch slave via execution of command on the Master

Launch command: java -jar /home/jenkins/slave.jar

Availability: Keep this slave on-line as much as possible

Node Properties

Environment variables

List of key-value pairs

name: BOARD_OVERLAY

value: boards/blue-pi.board

Delete

name: DISTRIB

value: distribs/nologger.dist

Delete

Add

Save

Adding a toolchain

- Generic qemu ARM toolchain is pre-installed
- To install your own (overview):
  - Obtain or build your SDK
  - Install it inside the container in /userdata/toolchains
  - Modify /userdata/conf/tools.sh to reference it

Get SDK into the container

- To build the SDK in Yocto Project:
  - Inside your yocto build directory:
    - bitbake <*image-name*> -c do_populate_sdk
    - docker ps *(note the container id)*
    - docker cp tmp/deploy/sdk/poky-*.sh <*container-id*>:/tmp
- Install the SDK in the container:
  - At the shell inside the container:
    - /tmp/poky-....sh
      - (specify an installation path under /userdata/toolchains, like: /userdata/toolchains/poky/2.0.1)

Tell Fuego about SDK

- Add a new “xxx-tools.sh” file for this toolchain
  - Determine a platform name (e.g. foo)
  - Create file: /userdata/toolchains/xxx-tools.sh
    - e.g. /userdata/toolchains/foo-tools.sh
  - Export variables needed by the toolchain in the file
    - e.g. PREFIX, ARCH, CC, AS, LD, etc.
  - Can source a Yocto Project environment_setup script
    - In this case, set SDKROOT variable
  - See qemu-armv7hf-tools.sh and lager-tools.sh for examples
- Set PLATFORM environment variable in board file
  - e.g. PLATFORM=“foo”

Adding a test - overview

- A Fuego test consists of:
  - Actual test program (the thing that runs on the target)
    - Shipped as source
  - Test shell script
  - Results parser script (for benchmarks)
  - Results evaluator expression (for benchmarks)
  - Jenkins test declaration
- Test can be Functional or Benchmark

Functional tests

- Detects regressions
- Result is pass/fail
- Stress tests are defined as functional tests

Benchmark tests

- Integrated plotting
- Parser to obtain value from test log
- Specification for data name and threshold for pass/fail

Test program

- Usually a pre-existing, compiled test program
- Source and patches are shipped in fuego-core repository
- Is cross-compiled by fuego for each target
- Can use one already in your distribution
  - Use ‘is_on_target()’ function to locate it

Test script

- Shell script describes how to:
  - Build the test program (if applicable)
  - Deploy the test to the target
  - Execute the test on target, and collect results
  - Test for success or failure, by examining the log
- Can define the following functions:
  - test_pre_check, test_build, test_deploy, test_run, test_processing
- Include a fuego engine script
- Script calls fuego functions to perform operations with the target

Fuego functions

- Fuego functions available in test scripts:
  - put/get – transfer files to/from target
  - cmd – execute command on target
  - report – execute command, and put results in log
  - log_compare – check log for a pattern, to check for pass or fail
  - hd_test_mount_prepare – mount a filesystem for a test
  - hd_test_clean_umount – unmount a filesystem after a test
- There are more
  - See examples in other scripts and wiki page:
    - http://bird.org/fuego/Test_Script_APIs

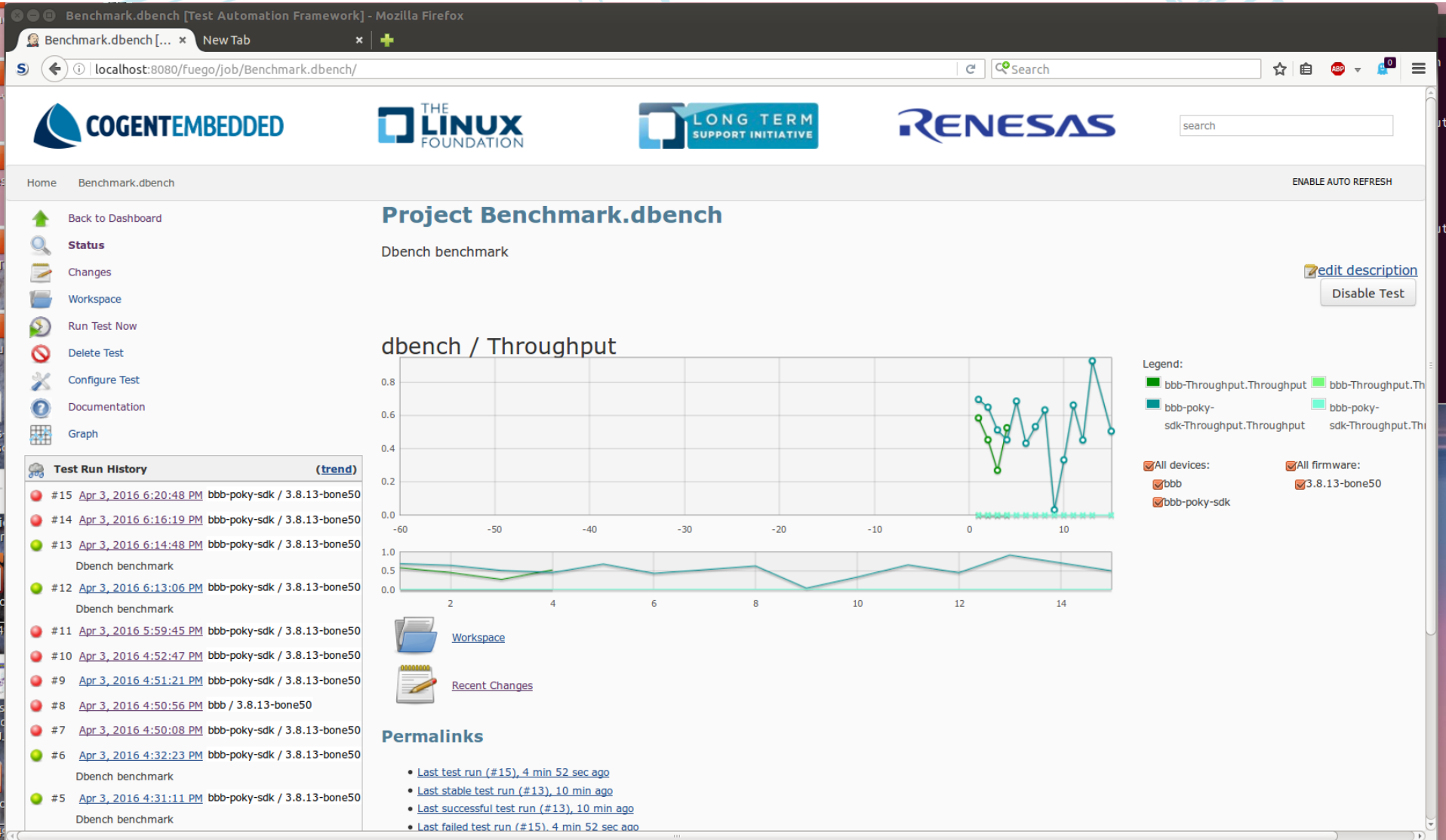
Shell script example

```
tarball=synctest.tar.gz

function test_build {
    make && touch test_suite_ready || build_error "error while building  test"
}

function test_deploy {
        put synctest  $FUEGO_HOME/fuego.$TESTDIR/
}

function test_run {
    assert_define FUNCTIONAL_SYNCTEST_MOUNT_BLOCKDEV
    assert_define FUNCTIONAL_SYNCTEST_MOUNT_POINT
    assert_define FUNCTIONAL_SYNCTEST_LEN
    assert_define FUNCTIONAL_SYNCTEST_LOOP

    hd_test_mount_prepare $FUNCTIONAL_SYNCTEST_MOUNT_BLOCKDEV \
        $FUNCTIONAL_SYNCTEST_MOUNT_POINT
    report "cd $FUNCTIONAL_SYNCTEST_MOUNT_POINT/fuego.\
        $TESTDIR; $FUEGO_HOME/fuego.$TESTDIR/synctest  \
        $FUNCTIONAL_SYNCTEST_LEN  \
        $FUNCTIONAL_SYNCTEST_LOOP"

    hd_test_clean_umount $FUNCTIONAL_SYNCTEST_MOUNT_BLOCKDEV \
        $FUNCTIONAL_SYNCTEST_MOUNT_POINT
}

function test_processing {
        log_compare "$TESTDIR" "1" "PASS : sync interrupted" "p"
}

. $FUEGO_SCRIPTS_PATH/functional.sh
```

Benchmark extras

- Extra files for plotting benchmark data
  - Parser.py, reference.log and tests.info
- Parse the test results (parser.py)
  - Extract data from the log, using a regular expression, and format it into a python map
- Specify threshold for pass/fail (reference.log)
  - Put an expression in reference.log file
- Indicate the variable(s) to plot (tests.info)
  - Global file tests.info has lines for all tests' plottable variable

Plot example



Outline

Introduction
Architecture
Customization
Vision

Vision

- (See next presentation)

Resources

- Wiki: http://bird.org/fuego/FrontPage
- http://bird.org/fuego/Fuego_Quickstart_Guide
- Mail list:
  - https://lists.linuxfoundation.org/mailman/listinfo/fuego
  - Mail to: fuego@lists.linuxfoundation.org

CE Workgroup

Fuego
Status and Future Directions

Tim Bird

Architecture Group Chair

LF CE Workgroup

Outline

- Vision
- Recent activity
- Future directions

Vision

Vision

Do for testing
what open source has done
for coding

Testing problem

- Much of testing is ad-hoc
  - Custom in-house work
  - LOTS of manual steps
- There are open source test programs
  - Can collaborate on these as coding projects
    - LTP, bonnie, cyclictest, netperf, …
  - Test frameworks have parts of the puzzle:
    - Jenkins, LAVA, KernelCI
- Key pieces are left to the user:
  - What tests to run
  - How to run the test programs
    - How to customize tests for different scenarios
    - How to automate tests
  - How to interpret the results

Solution

- *Reduce duplication of effort in testing*
- Allow developers and testers to share effort that each company is doing by itself
- Strategy:
  - Identify manual steps
  - Capture them
  - Create ways to share “test collateral”
- Test framework has to gain popularity to create “community effect”
- Focus on real tests and test results

Required features

- Allow quick and easy setup
- Support a wide variety of configurations and build systems
  - Yocto Project/OE, Buildroot, etc.
- Support a wide variety of targets
- Support a wide variety of connection types:
  - serial, ssh, adb, ttc
- Make it easy to create and publish new tests

Vision of sharing

- Need to share test experience and collateral
- Not just test programs
  - Test results based on boards, distros, hardware
  - Parsing methods
  - Test parameters
- “Test app store”
  - Thousands of tests to choose from
- Results from tens of thousands of test nodes
  - Crowdsourced results (e.g. Wikipedia)

Increased automation

- Need heterogeneous, multi-node testing
  - Test environment is more than just the Linux distribution on the machine, with local hardware
  - Lots of tests need other endpoints or external hardware to communicate with
    - USB, i2c, Ethernet, wifi, canbus, video/audio inputs and outputs
  - These are the hardest tests to automate
    - Require specialized hardware or configuration
      - E.g. USB switcher, CANbus packet injector
- Make it possible to share these rare setups

Increased hardware coverage

- Big problem in Linux kernel community is testing on different hardware, to ensure things don't break as patches are accepted upstream
- Most successful board-level project is KernelCI
  - 10 labs, 160 boards, 2 million boots
- Want to do same thing, but with individual boards
  - Not necessarily kernel testing

Sharing hardware

- Ten thousand test nodes
- Ability to run tests at request of community
  - Ability to customize a test to find individual bugs
  - Example workflow:
    - User reports a bug on hardware xyz
    - Developer runs test on node 2374 to reproduce
- Requires granting access to 3rd parties
- Requires trust network
  - Use same mechanisms as kernel:
    - Traceable affidavit
    - Signing
    - Validation

Recent Activity

Recent Activity

- Survey of test tools
- Collection of test stories
- Recent Fuego work
- Branding
- Infrastructure

Other test tools

- LAVA
- KernelCI
- KSelftest

LAVA

- Linaro's test framework
- Is a very powerful test scheduler
  - Understands how to interact with many boards and bootloaders
  - Has a very scalable, secure architecture
- Does not include tests themselves
- Is pretty darn complex
- Is in midst of version change (to 2.0)
- Used by AGL-JTA (Fuego precursor for Linux Foundation automotive group) for board management

KernelCI

- Project by Linaro specifically designed to find kernel boot regressions
  - 10 labs, 160 boards, 2 million boots
  - Testing many upstream source trees
- Centralized management of test system
- Have good support for board farms (via LAVA)
- Have results aggregation and comparison
- Has track record of actual fixes based on bugs found (~165)
- Focus only on boot testing for now

KSelftest

- Kernel unit test framework
- Has tests for individual kernel sub-systems
- Has no automation
  - No output consistency
    - Requires a human to interpret output of each test
  - No notion of multi-node testing that I’m aware of
- Is relatively new
  - Kernel version 4.1 can install test on target

Test tools conclusion

- Test frameworks focus on different parts of the overall test picture
  - None focused on abstracting test invocation and analysis
- Should figure out how to collaborate on common pieces
- Make standards so that all tools, labs, tests, benefit from improvements

Test Stories

- Collected test stories at Linaro Connect and ELC Europe
- May collect more at future events
- Haven’t posted on wiki yet…

Results of “test stories” survey

- Everyone wants to test something different
- Lots of manual activities
- Hard to convert from manual to automated
  - Every board and configuration has quirks
  - Need to control external entities during test
    - Heterogenous multi-node
      - eg canbus, wifi, Ethernet, video input, i2c devices, etc.
- No sharing of test collateral
  - How to share “test expertise”?

Recent Fuego work

- test_pre_check
- “ftc” tool
- Documentation
- Proxy support

test_pre_check()

- New optional base script function to support pre-testing the target and environment
- Added is_on_target() helper routine
  - To detect binaries on the target
  - Avoid building test program if it's already in the distribution
- Plan to move ASSERT_DEFINES into test_pre_check() function
- Abort the test early if target or environment is missing key feature
- Maybe give a different error
  - E.g. 'configuration problem' or 'unmet dependencies'

‘ftc’ tool

- Ability to add, update test variables in board file
- Test can store persistent information about a board
- Purpose is to support target probing and saving of found information
- Ability to launch a test from command line
- Ability to query targets and tests

Documentation

- Test APIs
  - Have templates for test API documentation
  - Have about 50% of functions documented
- More docs about fuego details:
  - phases, logs, environment variables, etc.

Proxy support

- Allow for installation inside a corporate firewall
- Patches submitted by Daniel Sangorrini
  - Still processing them (sorry Daniel)

Branding

- Name change from JTA in March
- Official logo:
  - Red, bold, Arial, slightly rotated:

Fuego

  - Vertical flames:

- Official candy:
  - Hot Tamales
    - Spicy cinnamon

What is “Fuego”?

- Fuego = Tierra del Fuego - one of the places on earth where penguins live
- Fuego = Fire – often associated with trials and purifying
- Fuego – it sounds neat

Fuego

It’s hot!

Infrastructure

- Web site (wiki)
  - http://bird.org/fuego/FrontPage
- Mail list:
  - https://lists.linuxfoundation.org/mailman/listinfo/fuego
  - Mail to: fuego@lists.linuxfoundation.org
- Virtual private server
  - Purpose is for online demos, and for people to try out the interface, without installing the tool
  - Not populated yet
    - Working on qemu issues on VPS

Future Directions

Future directions

- Easier installation and setup maintenance
- Target dictionary
- Test packages
- Test interface standards
- Other stuff
- What Fuego is not…

Easier installation

- Tools to assist installation
  - Provide “myboard” and set a few params (and maybe rename it) with the command line tool
  - No editing of files!
- Provide a container on docker hub to eliminate install phase completely, for most users
- Health check test

“Target dictionary”

- Defined place for per-board test parameters and collateral
- Re-organization of test specs and test plans
- Ties in to sharing test collateral
  - E.g. if someone has stuff working on a beagle-board, they should share their results
- ‘ftc’ tool is precursor to this

Test packages

- Make tests separate from the framework
- Define what needs to be shared
  - Collect materials that are scattered all over in the repository: test script, source, jenkins config, test specs, test plans, log parser
- Test plugin architecture
- Tool to manage packages:
  - Ability to install individual test
  - Create a package from existing materials
  - Publish a test package
- Import via package or git repository

Test interfaces standards

- Allow multiple front-ends and back-ends
- Board control interface
- Hardware and firmware recommendations
  - Eg “expose a serial console”, “support tftp boot”
- Compliance tests and ratings

Other stuff

- Self-tests for the test framework
- Matrix of tests vs. board results
- More tests
  - Refine the board bringup tests from Renesas
  - kselftest
  - Kernelci – kernel boot test
    - Waiting for serial console support
- Move official docs to ascii-doc (or some other markdown)

Other stuff (cont.)

- De-clutter the Jenkins front end

- Improve documentation (more)
- Handle USB connections
  - For ADB-based targets
  - For Sony debug board
- Support for boards with only a serial console
  - Have contractor for this work (Lineo Solutions)

What Fuego is NOT

- Board farm tool
  - Can support multiple nodes, but that's not the focus
  - Handling the scalability of board farms is difficult and requires extra hardware
  - Focus on a single developer testing a single board
    - To scale out, make more host nodes
- Test results aggregator (yet)
  - This will come in time. Focus for now is on scaling out the actual tests

Stuff deferred

- Send data to centralized repository
- Make it possible to join a decentralized test network
  - Help solve the “developer can’t test on different harware” problem

Thanks