Fuego 1.5 Features Status

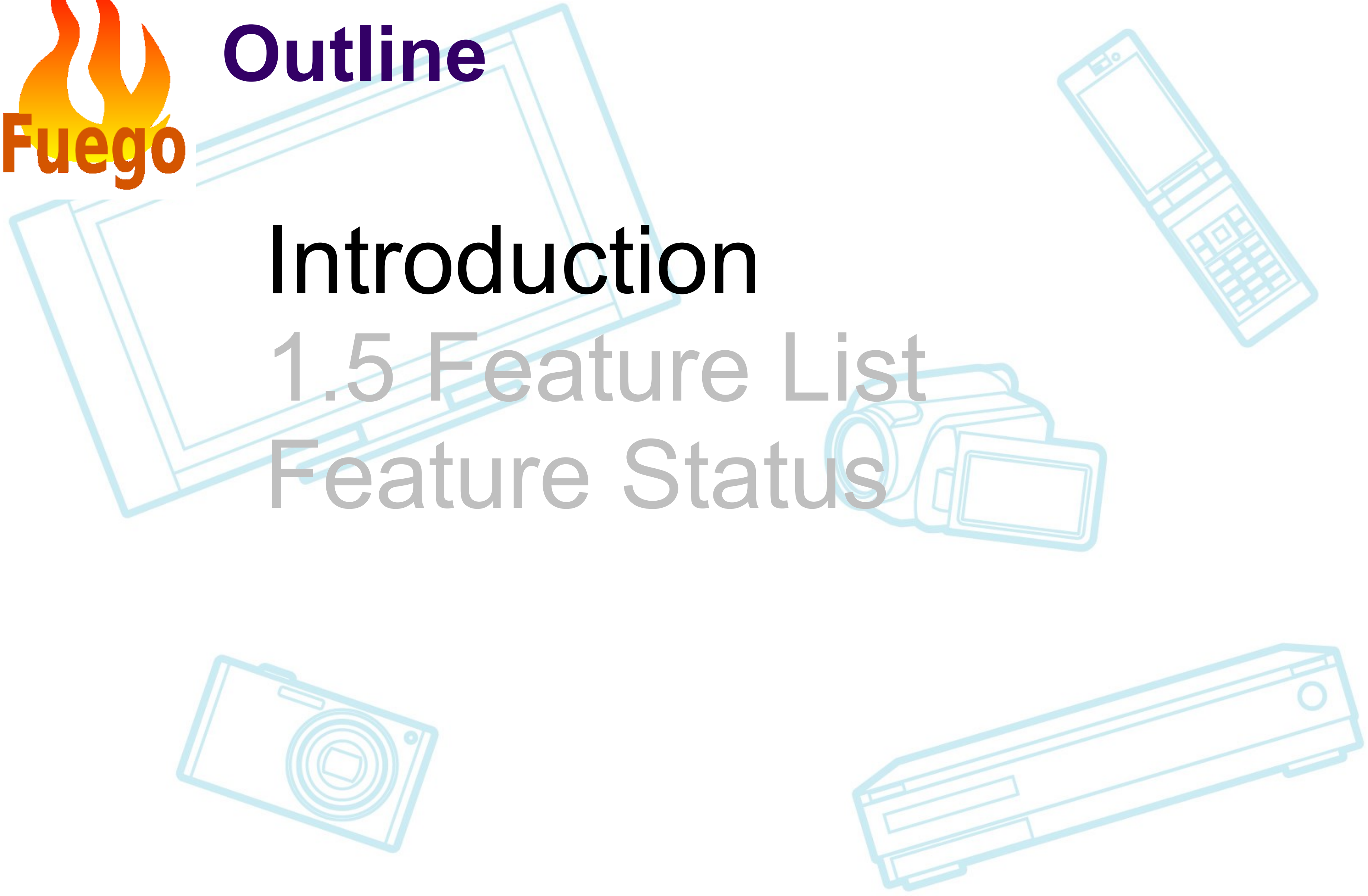
July 2019

Tim Bird

Fuego Maintainer

Sony Electronics

Outline



Introduction

1.5 Feature List

Feature Status

Introduction

- Fuego v1.4 was release in January 2019
- Fuego v1.5 was originally intended to be a quick, small release
  - Focus:
    - simplify and enhance install
    - re-organize directory structure
  - It grew in a lot of directions
- 1.5 release is very close to completion

Exiting core feature overview

- Distribution of Linux for testing
- Build system
  - Architecture-neutral
  - Inherently cross-platform
- Collection of tests
  - Scripts for test execution
  - Results parsing, analysis, and visualization
- Host/target oriented
  - Multiple transports
- Integrated Jenkins front end/back end
- ‘ftc’ command line tool

Outline

Introduction

1.5 Feature List

Feature Status

1.5 Feature List

- (internal) Simplified directory structure
- Upgraded base distribution
- Upgraded Jenkins version
- Jenkins-less install
- Install without container
- New default Jenkins port (8090)
- New tests
- Batch tests
- ftc command line completion

Simplified directory structure

- Mostly for internal cleanliness
- Removed ‘engine’ directory
- Move ‘fuego-core’ inside ‘fuego’ directory
  - Now have only a single top-level directory
  - ‘engine’ symlink left for backwards compatibility
  - Install now automatically downloads ‘fuego-core’
    - One less manual step during install

Upgraded base distribution

- Base of Fuego Linux distribution changed from Debian Jessie to Debian stretch
  - Jessie = Debian 8, released 2015-04
  - Stretch = Debian 9, released 2017-06
  - Next Debian, “Buster”, was just released
    - 2019-07
- Changed to ‘slim’ version of base distribution
  - Should save space on host used by docker images

Upgraded Jenkins version

- Fuego v1.4 used Jenkins version 2.32.1
- v1.5 upgraded to version 2.164.1
- Now using latest security updates
- Can use more recent plugins

Jenkins-less install

- Can build Fuego docker container without Jenkins
- Can now use Fuego “headless”
  - Jenkins is a very heavy-weight java app
  - Container is smaller
  - Use command line tools for Fuego operations
- Note:
  - Miss out on Jenkins triggers, test scheduling, results visualization

Install without a container

- Can install Fuego directly to a Debian host
- Does not build a Fuego docker container
- Use ‘install-debian.sh’
- Can be used for a light-weight installation of Fuego
  - e.g. directly onto a target
  - into a node in another framework (e.g. LAVA)
- Security Note:
  - Tests are run natively on the host (the host-side portion of the test)
  - Be very careful running tests from third parties

New default Jenkins port (8090)

- Old default port for Jenkins was 8080
  - Old url: http://localhost:8080/fuego
- New default port is 8090
  - New url: http://localhost:8090/fuego
- This avoids conflict between Fuego and existing Jenkins installation
  - Or some other service on port 8080
- Also, port is configurable during install:
  - ex: $ install.sh fuego 7777
  - You can continue to use port 8080 if you need to

New tests

- Functional.brctl
- Functional.iperf3_server
- Functional.ipmi
- Functional.libxml
- Functional.module_init_tools
- Functional.multipathd
- Functional.nscd
- Functional.openct
- Functional.openhpid
- Functional.vconfig

Batch tests

- Batch tests = a mechanism for running multiple Fuego tests in sequence
- Replaces ‘testplans’
- New ‘run_test’ function in core library
- testplan data was moved to fuego_test.sh

Creating a batch test

- Create a Functional test, with a base name prefix of “batch_”
  - ex: Functional.batch_filesystem_tests
- Put calls to “run_test” in fuego_test.sh test_run() function
- Put testplan json into fuego_test.sh
  - Defined at BATCH_TESTPLAN variable using very specific syntax
- Use a parser that understands nested TAP
  - Just copy parser from Functional.batch_default

Batch test example:

```
BATCH_TESTPLAN=$(cat <<END_TESTPLAN
{
    "testPlanName": "smoketest",
    "default_timeout": "6m",
    “tests”: [
        { "testName": "Functional.fuego_board_check" },
        { "testName": "Benchmark.hackbench" },
        { "testName": "Benchmark.netperf" },
    ]
}
END_TESTPLAN
)

function test_run {
    export FUEGO_BATCH_ID="st-$(allocate_next_batch_id)"

    # don't stop on test errors
    set +e
    log_this "echo \"batch_id=$FUEGO_BATCH_ID\"“
    run_test Functional.fuego_board_check
    run_test Benchmark.hackbench
    run_test Benchmark.netperf
    set -e
}
```

Using run_test() function

- Arguments are same as for ‘ftc run_test’
  - Can specify spec
  - Can specify timeout, reboot, cleanup flags, etc.
- The batch test should have a corresponding entry in the testplan for each test executed via run_test()
  - Specifying the same parameters, if possible

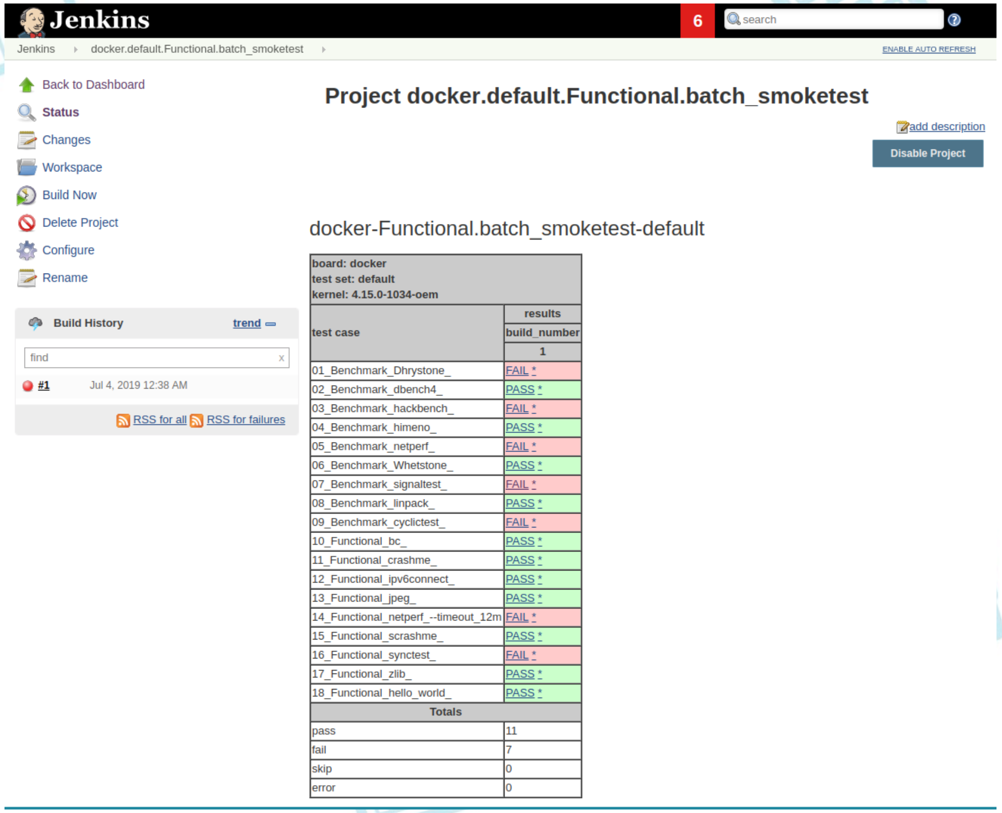
Using batch tests

- To install:
  - ftc add-jobs –b myboard –t Functional.batch_foo
  - Create a job for Functional.batch_foo
  - Also creates jobs for the child tests (found in the embedded testplan)
- To run:
  - Jenkins trigger job:
    - myboard.default.Functional.batch_foo

    or
  - ftc run_test –b myboard –t Functional.batch_foo

Batch test results (Jenkins)

- To view results in Jenkins
  - Jenkins examine myboard.default.Functional.batch_foo
  - Can click on new '*' link to navigate to sub-test page



Project docker.default.Functional.batch_smoketest

docker-Functional.batch_smoketest-default

board: docker
test set: default
kernel: 4.15.0-1034-oem

test case	results
	build_number
	1
01_Benchmark_Dhrystone_	FAIL *
02_Benchmark_dbench4_	PASS *
03_Benchmark_hackbench_	FAIL *
04_Benchmark_himeno_	PASS *
05_Benchmark_netperf_	FAIL *
06_Benchmark_Whetstone_	PASS *
07_Benchmark_signaltest_	FAIL *
08_Benchmark_linpack_	PASS *
09_Benchmark_cyclictest_	FAIL *
10_Functional_bc_	PASS *
11_Functional_crashme_	PASS *
12_Functional_ipv6connect_	PASS *
13_Functional_jpeg_	PASS *
14_Functional_netperf_--timeout_12m	FAIL *
15_Functional_scrashme_	PASS *
16_Functional_synctest_	FAIL *
17_Functional_zlib_	PASS *
18_Functional_hello_world_	PASS *
Totals	
pass	11
fail	7
skip	0
error	0

Batch test results (command line)

- Use the batch id to get results for a particular batch
- Find the batch id:
  - ftc gen-report –where test=batch_foo –fields timestamp,tguid,batch_id
- Single out data from a particular run
  - ftc gen-report –where test=batch_foo,batch_id=foo-7

Batch test notes

- Added batch_id field to run.json
- Can query using batch_id
  - ex: ftc gen-report –where batch_id=foo-12
- run_test uses TAP output format, but...
  - I had to extend the TAP format to deal with nested test output
    - Some sub-tests use TAP output format
  - I added a “[[batch_id]]” prefix to each line to allow the parser to find correct TAP lines
- NOTE:
  - kselftest has the same issue, but used a different solution
  - Maybe TAP needs to be extended

ftc command line completion

- Can use ‘ftc’ and use TAB to complete arguments
- Fuego provides a bash auto-completion script
- To use:
  - Type part of a command or argument, press TAB, and bash will provide a list of legal alternatives
  - e.g. ftc run-test –b be<TAB>
  - bash will complete the board name
    - ‘ftc run-test –b beaglebone’
- Very handy for manual operation

Prototype features in 1.5

- Support for tests from other frameworks
- Configurable back end (Squad)
- fserver support

Support for tests from other frameworks

- Functional.Linaro
- Functional.ptest

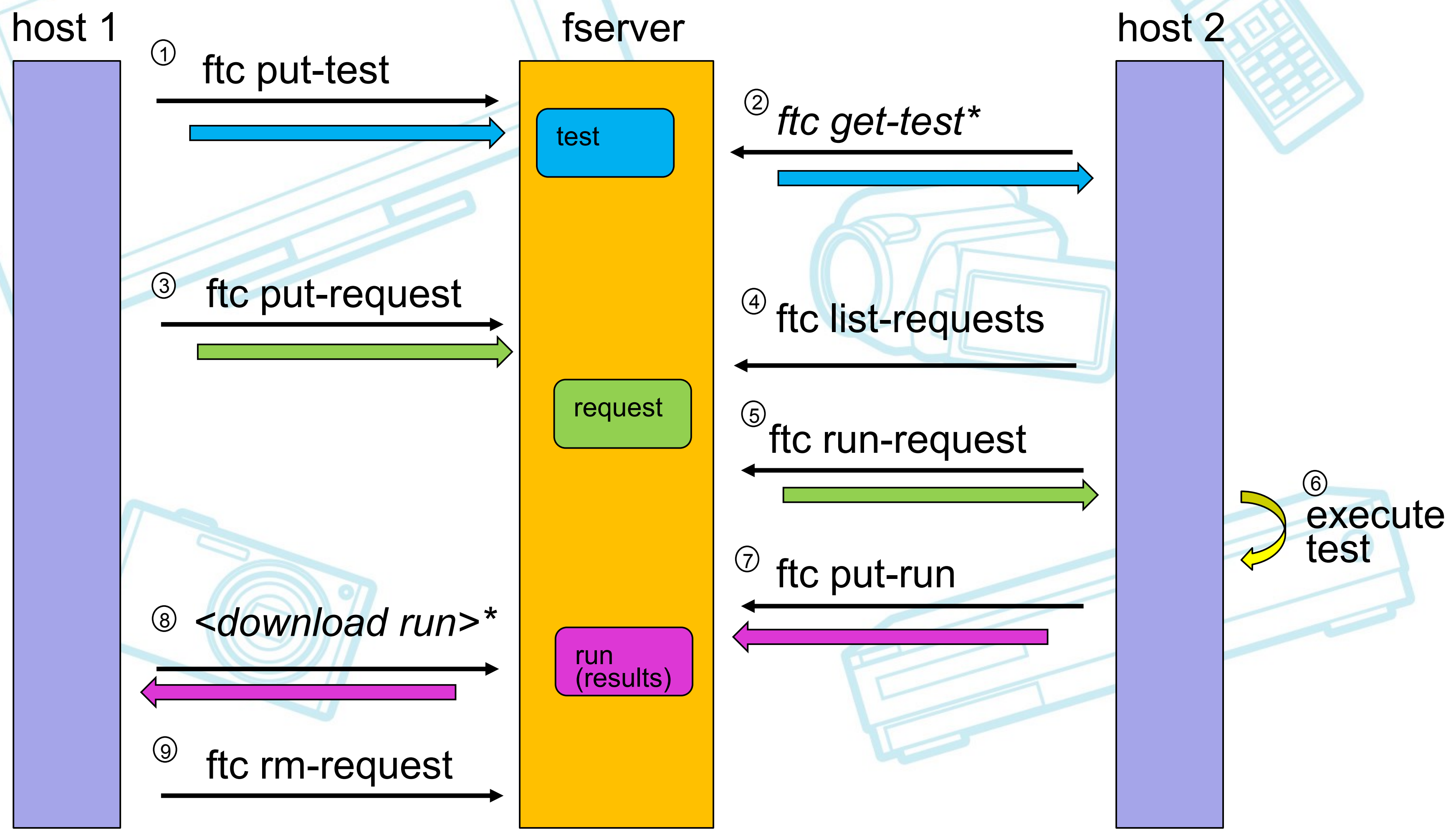
Configurable back end (Squad)

- Daniel will show this

fserver support

- fserver is a test object server
  - Can store tests, test requests, runs (results)
  - Can be used to deliver requests from one host to another, and return the results to the requesting host
  - intended to support distributed operation
- Is not complete
  - Needs more support in ‘ftc’
  - Needs to store more objects:
    - hosts, boards, target packages, image (build artifact)
- See http://fuegotest.org/wiki/Using_Fuego_with_fserver

fserver request flow



* = not written yet

fserver notes

- thin arrow in diagram is request
  - Note that all requests initiate at hosts (fserver never initiates a connection
    - The connection model will work from inside corporate firewalls
    - fserver can be put on port 80, and even target boards can access material from it (could put target packages on fserver, and run fuego core natively on a board)
- thick arrow in diagram is data flow
  - blue=test, green=request, magenta=run data

Resources

- For feature details and documentation see http://fuegotest.org/wiki/Release_1.5_Notes

Fuego