Test Driven Development: Unit-testing for the development of a Correlator-Beamformer

Mpho 'mm-Poh' Mphego

Test & Verification Engineer

SKA SA

mmphego@ska.ac.za

science and technology

Department:
Science and Technology
REPUBLIC OF SOUTH AFRICA

NRF National Research Foundation

SKA AFRICA
SQUARE KILOMETRE ARRAY

meerkat
karoo array telescope

HartRAO

SKA
SQUARE KILOMETRE ARRAY

Overview

- What is this TDD, you speak of?
- Unit-Testing is not TDD!
- To TDD or not to TDD?
- What is being tested?
- Types of tests done?
- How we do these tests using TDD?
- Future work and improvements.
- Demo, if time allows.

What is this TDD, you speak of?

- Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle: Requirements are turned into very specific test cases, then the software is improved to pass the new tests, only. [1]

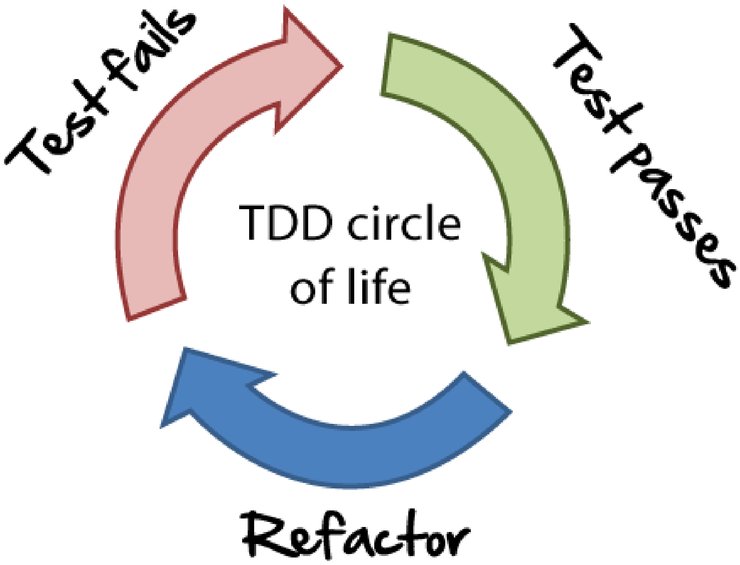
[1] Test Driven Development *[https://en.wikipedia.org/wiki/Test-driven_development]*

"Developer testing is an important step towards accountability. It gives developers a way to demonstrate the quality of the software they produce." - Kent Beck

Simplified Version

Test driven development in a nutshell (Red-Green-Refactor)**:**

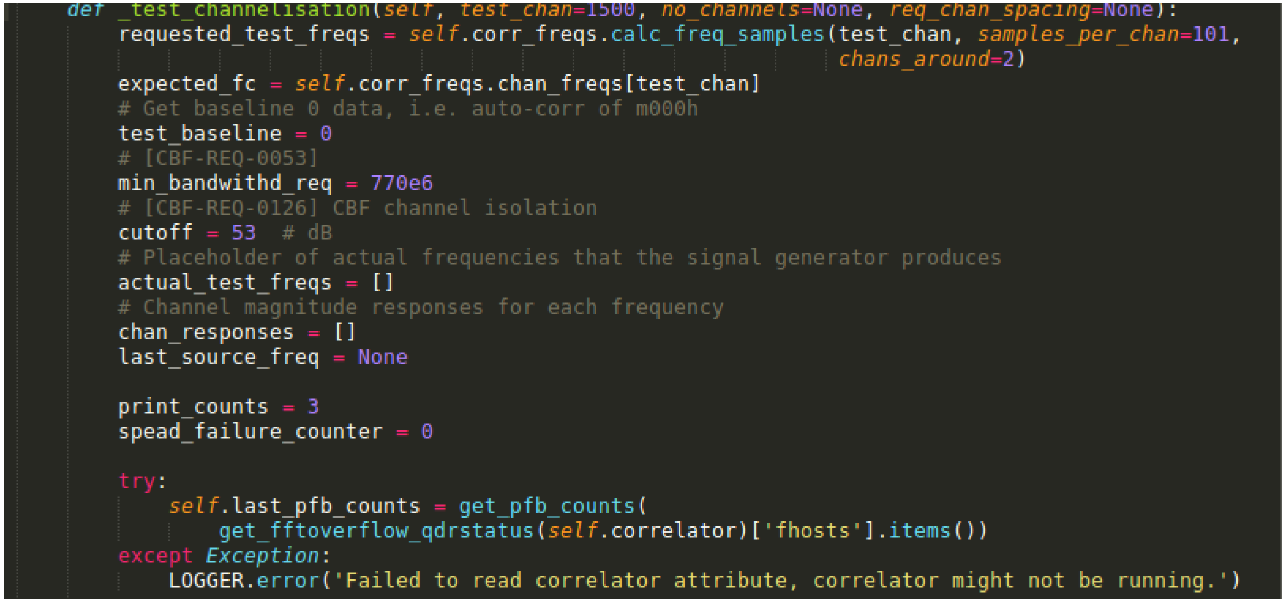
1. **Decide what the test will do**
2. **Write the test code**
3. **Watch the test fail**
4. **Write test logic as simple as possible**
5. **Pass the test**
6. **Refactor, removing any duplicates**
7. **Go back to 1**



Unit Testing is not TDD

Q: Is test driven development a form of unit testing?

A: TDD is a design methodology, it creates unit tests and forces you to make certain design decisions, usually improving the overall design

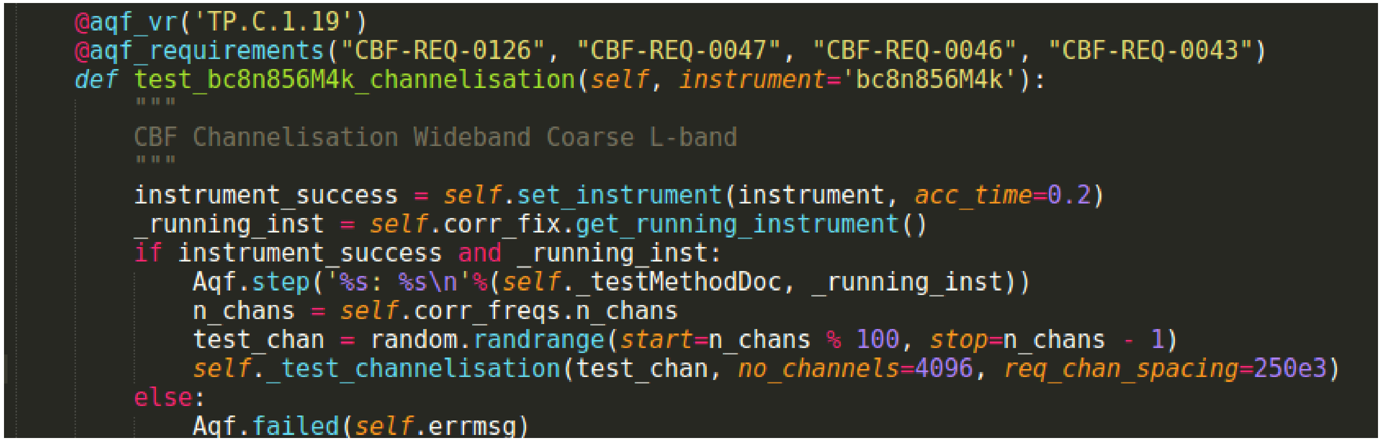


```
    def _test_channelisation(self, test_chan=1500, no_channels=None, req_chan_spacing=None):
        requested_test_freqs = self.corr_freqs.calc_freq_samples(test_chan, samples_per_chan=101,
                                                                 chans_around=2)
        expected_fc = self.corr_freqs.chan_freqs[test_chan]
        # Get baseline 0 data, i.e. auto-corr of m000h
        test_baseline = 0
        # [CBF-REQ-0053]
        min_bandwithd_req = 770e6
        # [CBF-REQ-0126] CBF channel isolation
        cutoff = 53  # dB
        # Placeholder of actual frequencies that the signal generator produces
        actual_test_freqs = []
        # Channel magnitude responses for each frequency
        chan_responses = []
        last_source_freq = None

        print_counts = 3
        spead_failure_counter = 0

        try:
            self.last_pfb_counts = get_pfb_counts(
                get_fftoverflow_qdrstatus(self.correlator)['fhosts'].items())
        except Exception:
            LOGGER.error('Failed to read correlator attribute, correlator might not be running.')
```

Typical Example of a unit test method

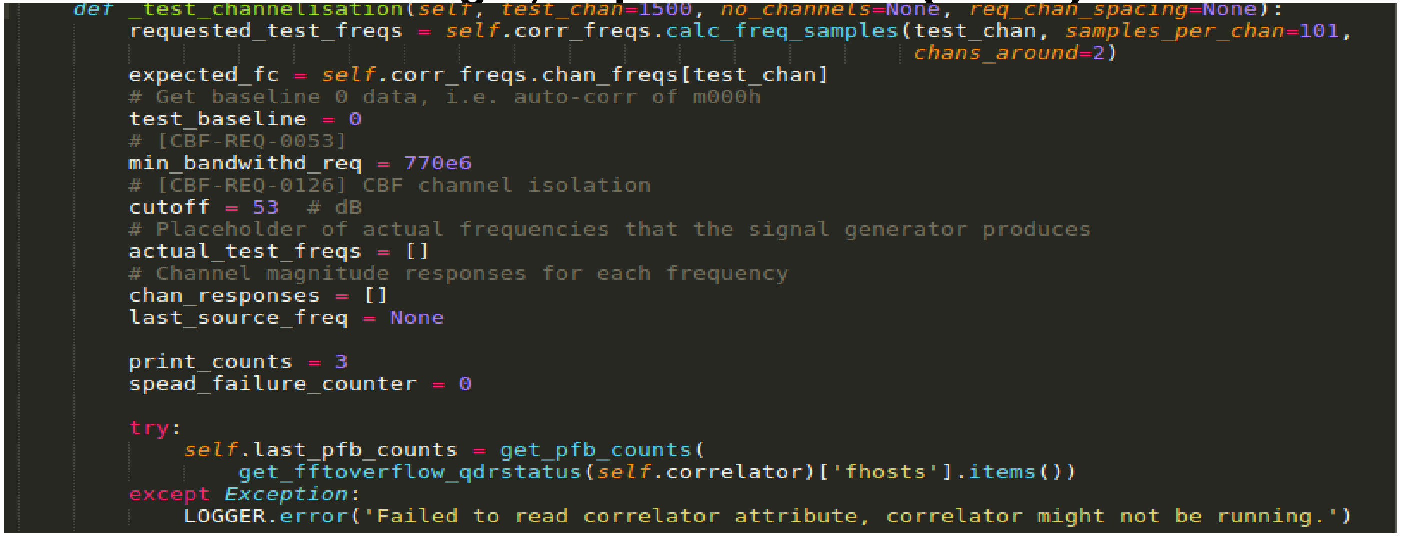
TDD = unit + test

1. Write the test first, run and ensure it fails (a Test)



```
    @aqf_vr('TP.C.1.19')
    @aqf_requirements("CBF-REQ-0126", "CBF-REQ-0047", "CBF-REQ-0046", "CBF-REQ-0043")
    def test_bc8n856M4k_channelisation(self, instrument='bc8n856M4k'):
        """
        CBF Channelisation Wideband Coarse L-band
        """
        instrument_success = self.set_instrument(instrument, acc_time=0.2)
        _running_inst = self.corr_fix.get_running_instrument()
        if instrument_success and _running_inst:
            Aqf.step('%s: %s\n'%(self._testMethodDoc, _running_inst))
            n_chans = self.corr_freqs.n_chans
            test_chan = random.randrange(start=n_chans % 100, stop=n_chans - 1)
            self._test_channelisation(test_chan, no_channels=4096, req_chan_spacing=250e3)
        else:
            Aqf.failed(self.errmsg)
```

2. Then write the logic, to pass the test (a Unit)



```
    def _test_channelisation(self, test_chan=1500, no_channels=None, req_chan_spacing=None):
        requested_test_freqs = self.corr_freqs.calc_freq_samples(test_chan, samples_per_chan=101,
                                                                 chans_around=2)
        expected_fc = self.corr_freqs.chan_freqs[test_chan]
        # Get baseline 0 data, i.e. auto-corr of m000h
        test_baseline = 0
        # [CBF-REQ-0053]
        min_bandwithd_req = 770e6
        # [CBF-REQ-0126] CBF channel isolation
        cutoff = 53  # dB
        # Placeholder of actual frequencies that the signal generator produces
        actual_test_freqs = []
        # Channel magnitude responses for each frequency
        chan_responses = []
        last_source_freq = None

        print_counts = 3
        spead_failure_counter = 0

        try:
            self.last_pfb_counts = get_pfb_counts(
                get_fftoverflow_qdrstatus(self.correlator)['fhosts'].items())
        except Exception:
            LOGGER.error('Failed to read correlator attribute, correlator might not be running.')
```

***The tests drive our development.

To TDD or not TDD?

You can do **unit testing** without doing **test driven development**. However you can't do **test driven development** without using **unit tests**.

When you do traditional **unit testing**, you write test **after** you wrote your code.

Test driven development approach is to write unit test **before** writing code.

Most interesting advantages of TDD (IMHO) comparing to simple Unit Testing:

Code is fully tested code upfront. It's painless testing.

It forces you to design your classes correctly.

It also forces you to keep it simple stupid.

The cycle of Red-Green-Refactor is the absolute procrastination killer!

Caveat

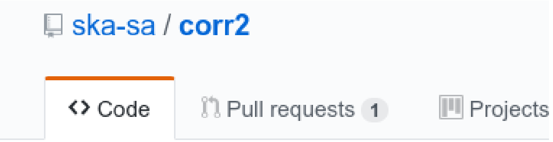
- The first time you do this it will take you a little bit longer before it’ll be faster.

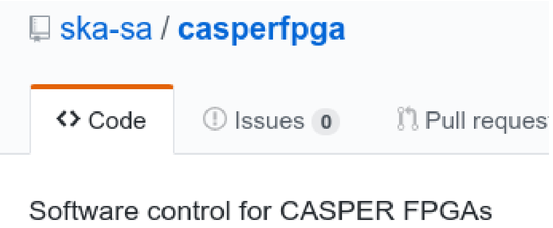
“One of the best programming skills you can have is knowing when to walk away for a while.” – Oscar Godson

What are we testing?

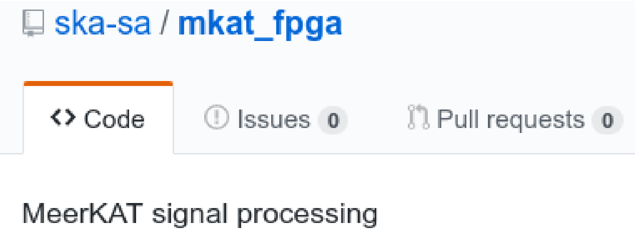
- Public Repos

CBF (Core) Maintained packages


ska-sa / corr2

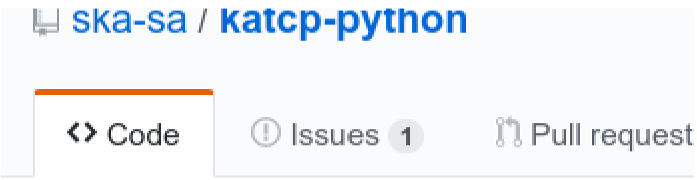

ska-sa / casperfpga

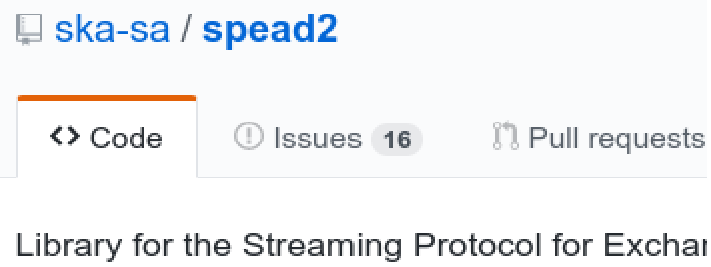
Software control for CASPER FPGAs


ska-sa / mkat_fpga

MeerKAT signal processing

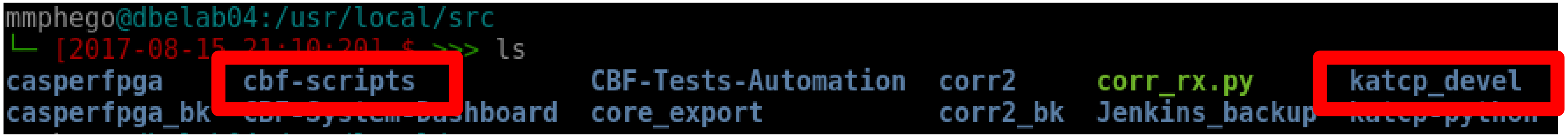
CBF dependencies


ska-sa / katcp-python


ska-sa / spead2

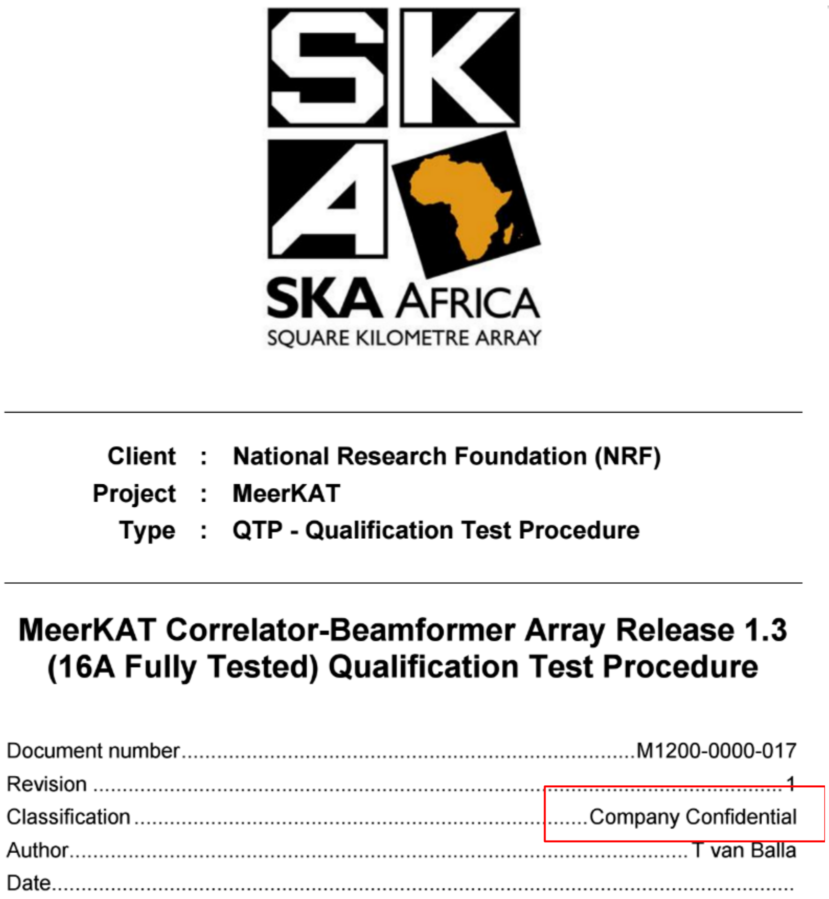
Library for the Streaming Protocol for Exchai

- Private Repos



```
mmphego@dbelab04:/usr/local/src
└ [2017-08-15 21:10:20] $ >>> ls
casperfpga     cbf-scripts           CBF-Tests-Automation  corr2     corr_rx.py      katcp_devel
casperfpga_bk  CBF-System-Dashboard  core_export           corr2_bk  Jenkins_backup  katcp-python
```

Types of tests we do?

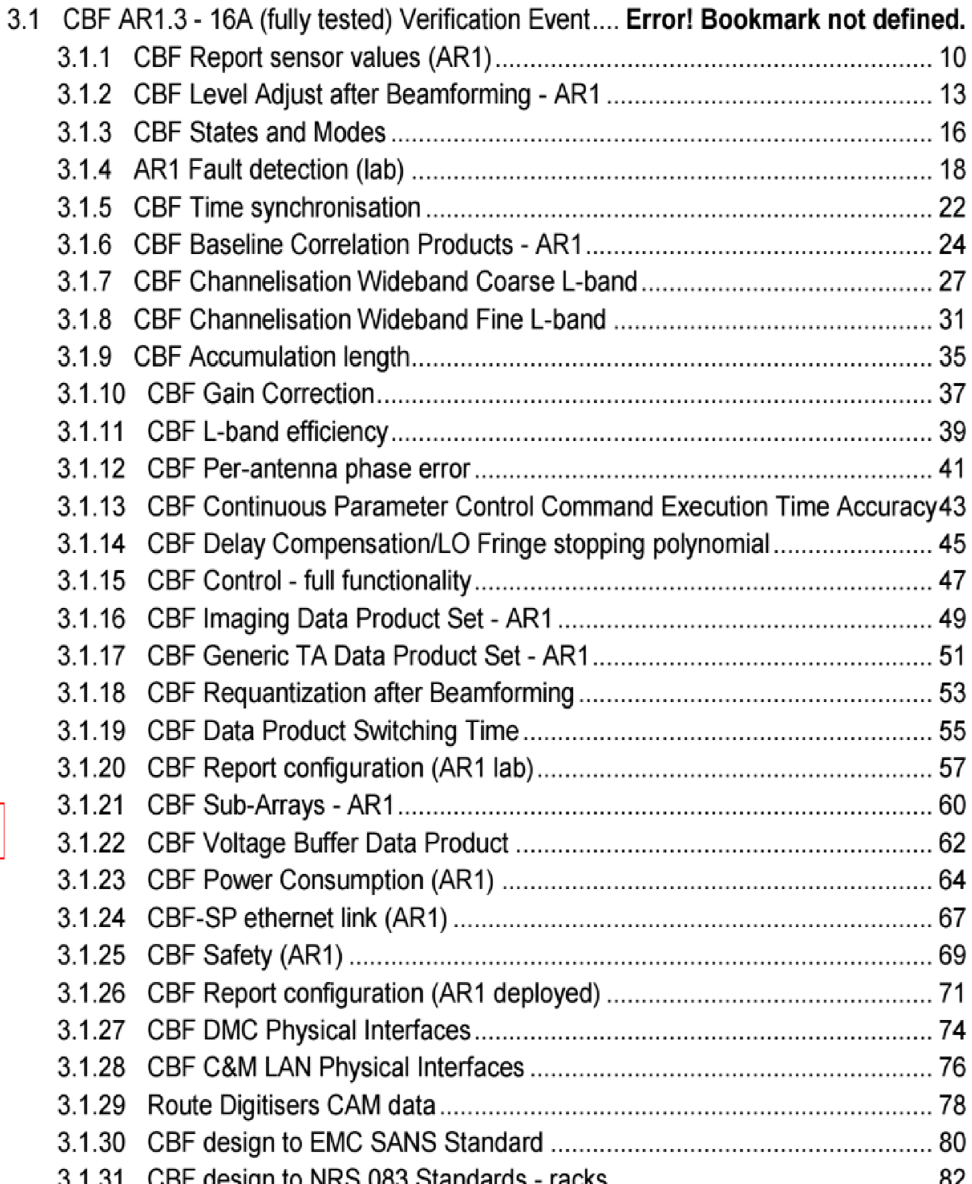


SKA AFRICA
SQUARE KILOMETRE ARRAY

Client : National Research Foundation (NRF)
Project : MeerKAT
Type : QTP - Qualification Test Procedure

MeerKAT Correlator-Beamformer Array Release 1.3 (16A Fully Tested) Qualification Test Procedure

Document number	M1200-0000-017
Revision	1
Classification	Company Confidential
Author	T van Balla
Date	



How is TDD applied to CBF Testing?

- Python Nose used as a unit-testing framework which makes testing easier.
- Nosetests extended with report generation plugin.
- Python decorators are associated with each tests, these are used in the generated report.
- Decorator specifies which verification requirements are covered in each test.

Non-hardcoded testing params

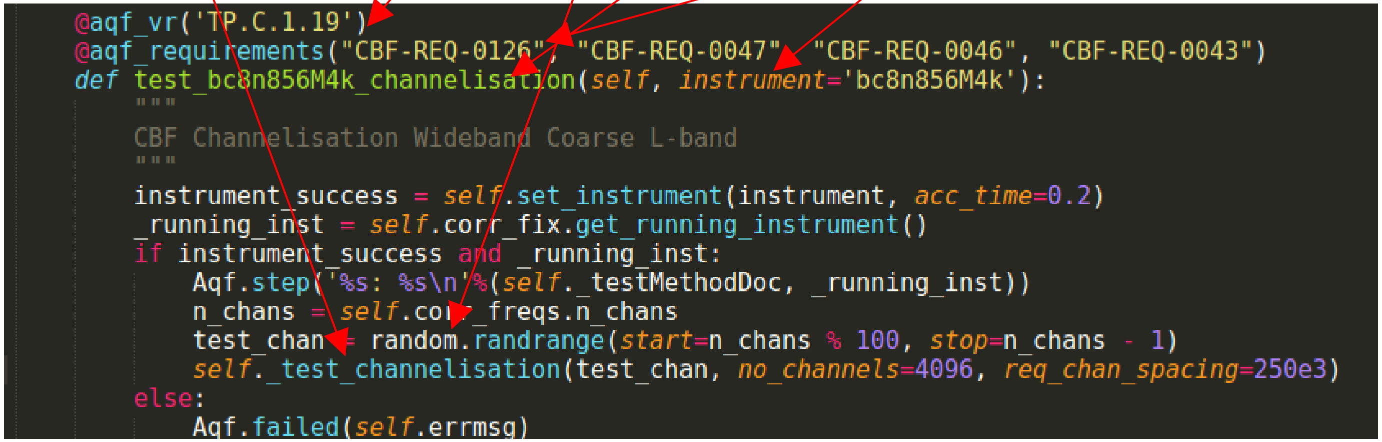
Actual test method

Test name

Instrument test is ran

CBF Verification Requirements

CBF Requirements as Req. Spec.

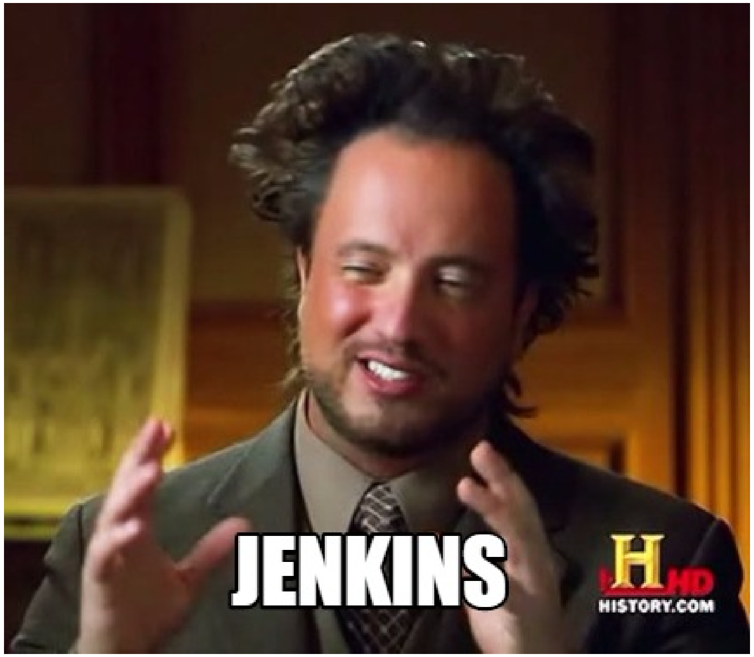


```
@aqf_vr('TP.C.1.19')
@aqf_requirements("CBF-REQ-0126", "CBF-REQ-0047", "CBF-REQ-0046", "CBF-REQ-0043")
def test_bc8n856M4k_channelisation(self, instrument='bc8n856M4k'):
    """
    CBF Channelisation Wideband Coarse L-band
    """
    instrument_success = self.set_instrument(instrument, acc_time=0.2)
    _running_inst = self.corr_fix.get_running_instrument()
    if instrument_success and _running_inst:
        Aqf.step('%s: %s\n'%(self._testMethodDoc, _running_inst))
        n_chans = self.corr_freqs.n_chans
        test_chan = random.randrange(start=n_chans % 100, stop=n_chans - 1)
        self._test_channelisation(test_chan, no_channels=4096, req_chan_spacing=250e3)
    else:
        Aqf.failed(self.errmsg)
```

Automated unit-testing with Jenkins CI

Manual to Automation:

- CBF Array Release 1(As mentioned by Francois) was qualified by means of using manual testing.
- CBF unofficial Array Release 1.5 which consisted of 16 Antenna 4k/32k correlator-beamformer was qualified with the use of Jenkins CI and reports are automagically generated
- Jenkins CI runs from a Docker container, Integrates with Git



*Jenkins CI: https://jenkins.io/

* Docker: https://www.docker.com/

What is Jenkins CI?

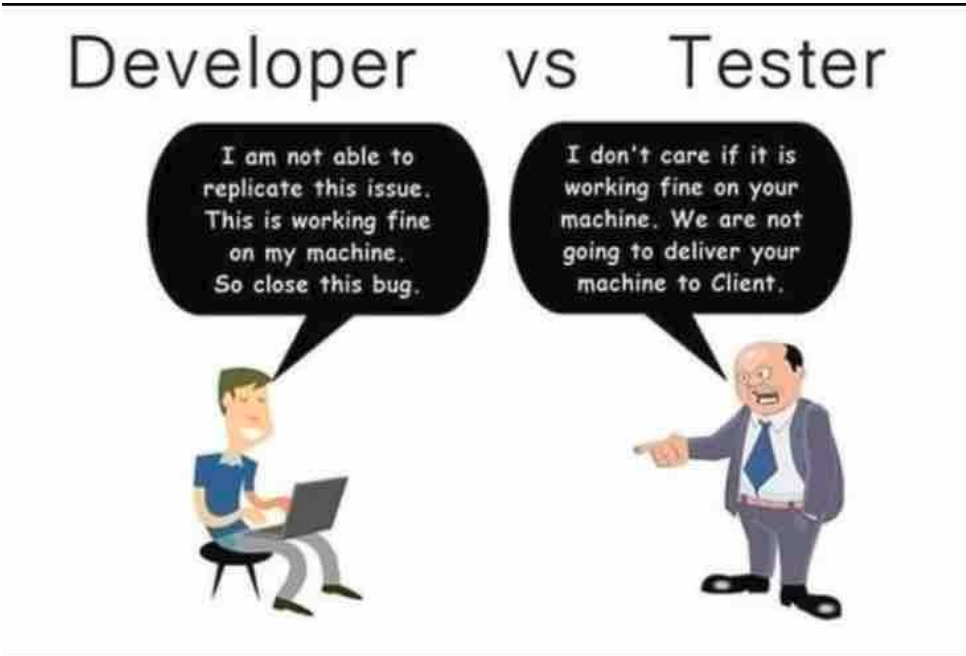
- Jenkins is a continuous integration and delivery application that builds and tests projects making it easier for developers to integrate changes to the project.
- Benefits: Team members integrate work frequently. Each integration is verified by an automated build to detect errors as quickly as possible.
- It is an auto-test platform which helps users to track where and when bugs are introduced.

*Jenkins CI: https://jenkins.io/

The feeling when you find a Bug!

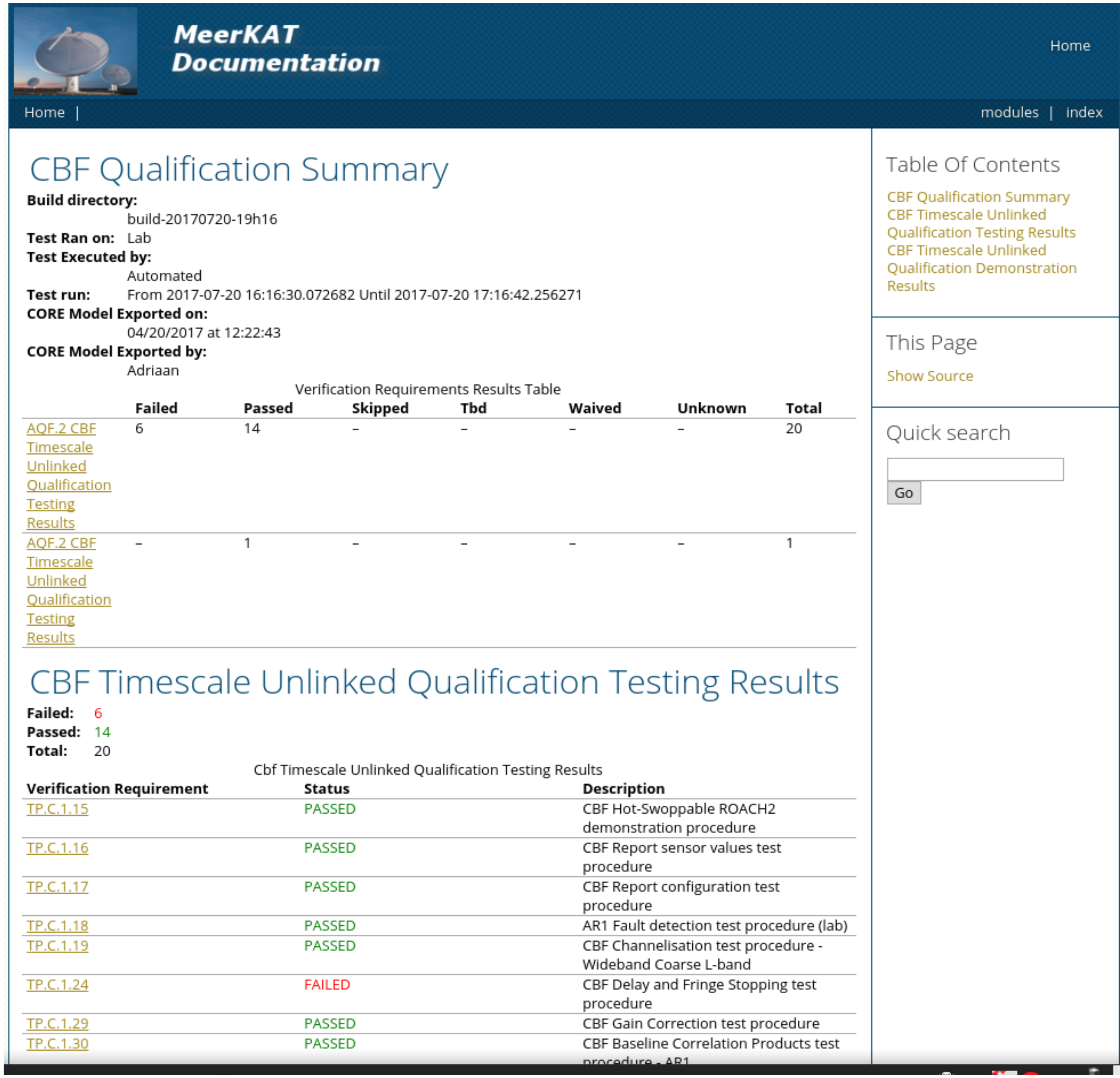
Top replies by *CBF Developers when Bugs are found!

- **AndrewM:** Well that’s weird, it has never done that before!
- **JasonM:** It must be a hardware problem!
- **PaulP:** You must probably have the wrong version of software installed!
- **PaulP:** Well, It works on my machine!
- **JasonM:** Did you install packages on correct path(s)!
- **JasonM:** There must be something funky happening with your data or maybe the switch is sending the data to wrong IP’s!
- **AndrewM:** I know it works, but I haven’t tested it!
- **JasonM:** Well, at least data is flowing!


Developer vs Tester
I am not able to replicate this issue. This is working fine on my machine. So close this bug.
I don't care if it is working fine on your machine. We are not going to deliver your machine to Client.

Automated Qualification Report

- Auto-generated using Python pocketsphinx library
- The requirement specs/req information is retrieved from an exported CORE xml model
- Pass/Fail statuses retrieved from *nosekatreport plugin for Python-Nose which creates a json file with all the relevant information



MeerKAT Documentation

Home

Home |

modules | index

CBF Qualification Summary

Build directory: build-20170720-19h16
Test Ran on: Lab
Test Executed by: Automated
Test run: From 2017-07-20 16:16:30.072682 Until 2017-07-20 17:16:42.256271
CORE Model Exported on: 04/20/2017 at 12:22:43
CORE Model Exported by: Adriaan

Verification Requirements Results Table

	Failed	Passed	Skipped	Tbd	Waived	Unknown	Total
AQF.2 CBF Timescale Unlinked Qualification Testing Results	6	14	–	–	–	–	20
AQF.2 CBF Timescale Unlinked Qualification Testing Results	–	1	–	–	–	–	1

CBF Timescale Unlinked Qualification Testing Results

Failed: 6
Passed: 14
Total: 20

Cbf Timescale Unlinked Qualification Testing Results

Verification Requirement	Status	Description
TP.C.1.15	PASSED	CBF Hot-Swoppable ROACH2 demonstration procedure
TP.C.1.16	PASSED	CBF Report sensor values test procedure
TP.C.1.17	PASSED	CBF Report configuration test procedure
TP.C.1.18	PASSED	AR1 Fault detection test procedure (lab)
TP.C.1.19	PASSED	CBF Channelisation test procedure - Wideband Coarse L-band
TP.C.1.24	FAILED	CBF Delay and Fringe Stopping test procedure
TP.C.1.29	PASSED	CBF Gain Correction test procedure
TP.C.1.30	PASSED	CBF Baseline Correlation Products test procedure - AR1

Table Of Contents

This Page

Show Source

Quick search

Go

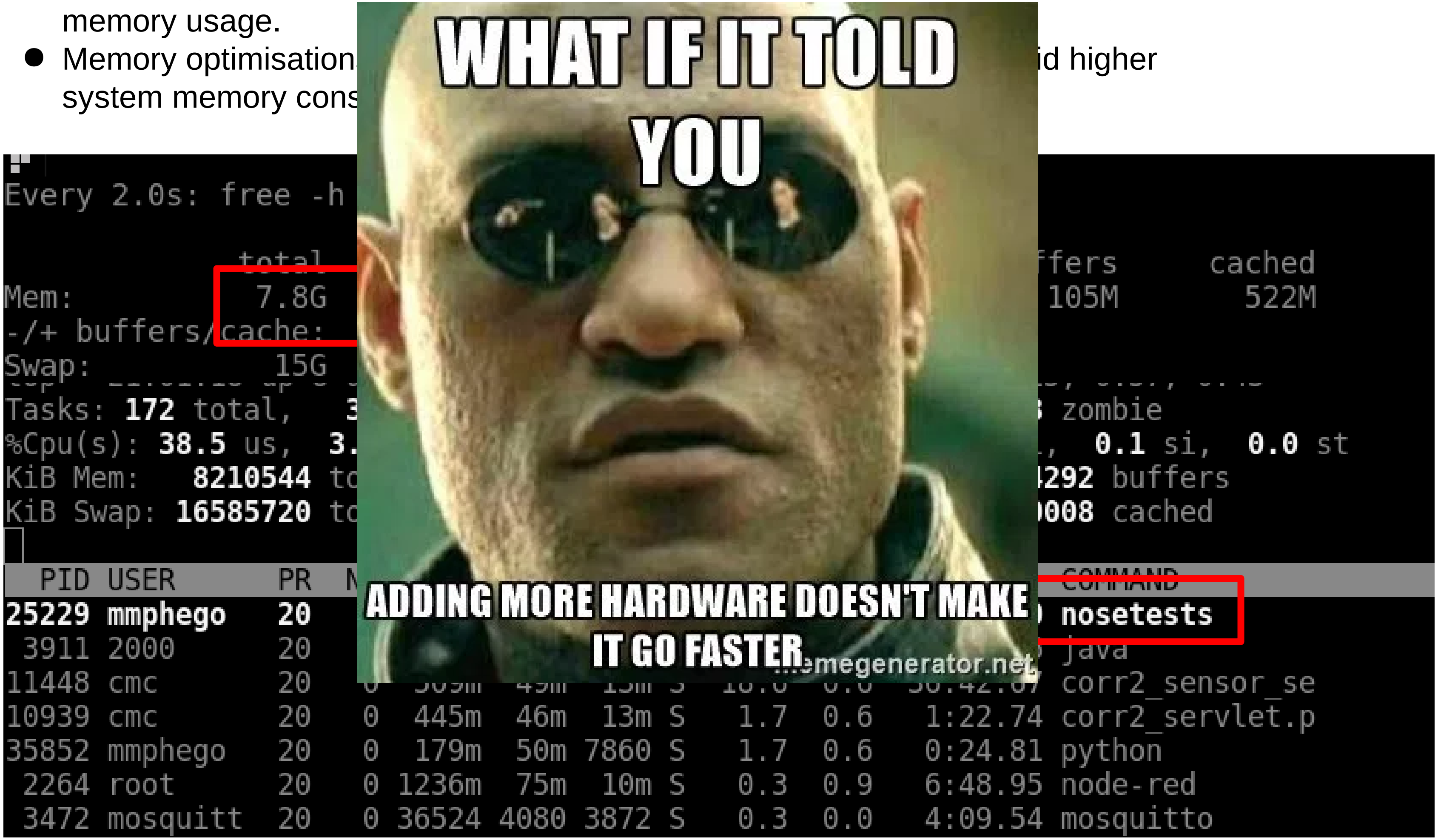
*https://github.com/ska-sa/nosekatreport

Future work and Improvements!

- Memory optimizations
- Git Hooks (automagically initiating a Jenkins build to run functional testing on all the changes made to repositories.)
- Auto bug reports with Jenkins-Jira and email notifications
- Latex integration: automagically created test qualification report
- CBF System Dashboard

Memory Optimizations

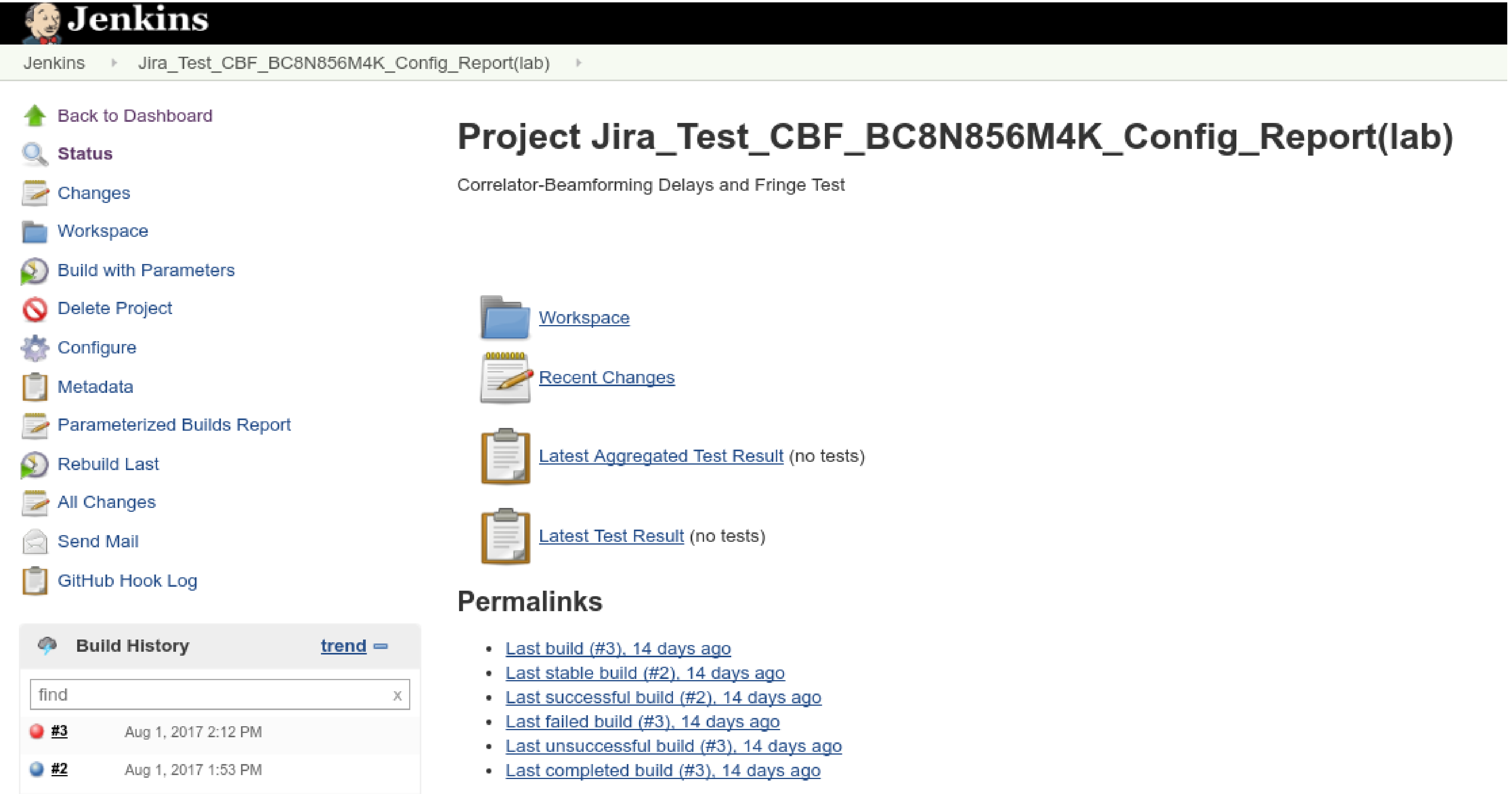
- Memory usage when running 4 Antenna 4k (CBF) test due to high data rate when retrieving correlated data.
- i.e. the higher the no. of antennas the higher the data rate vis-a-vi higher system memory usage.
- Memory optimisation id higher system memory cons



```
Every 2.0s: free -h

              total                                  ffers      cached
Mem:           7.8G                                   105M        522M
-/+ buffers/cache:
Swap:           15G
Tasks: 172 total,   3                                   zombie
%Cpu(s): 38.5 us,  3.                                 ,  0.1 si,  0.0 st
KiB Mem:   8210544 to                                  292 buffers
KiB Swap: 16585720 to                                  008 cached

  PID USER      PR  N                                  COMMAND
25229 mmphego   20                                     nosetests
 3911 2000      20                                     java
11448 cmc       20                                     corr2_sensor_se
10939 cmc       20   0  445m  46m  13m S   1.7  0.6   1:22.74 corr2_servlet.p
35852 mmphego   20   0  179m  50m 7860 S   1.7  0.6   0:24.81 python
 2264 root      20   0 1236m  75m  10m S   0.3  0.9   6:48.95 node-red
 3472 mosquitt  20   0 36524 4080 3872 S   0.3  0.0   4:09.54 mosquitto
```

Jenkins-Jira bug report API

- Integrating Jenkins CI with Atlassian Jira (if you haven't used it Google is your friend)



Jenkins

Jenkins ▸ Jira_Test_CBF_BC8N856M4K_Config_Report(lab) ▸

- Back to Dashboard
- Status
- Changes
- Workspace
- Build with Parameters
- Delete Project
- Configure
- Metadata
- Parameterized Builds Report
- Rebuild Last
- All Changes
- Send Mail
- GitHub Hook Log

Build History trend

find x

#3	Aug 1, 2017 2:12 PM
#2	Aug 1, 2017 1:53 PM

Project Jira_Test_CBF_BC8N856M4K_Config_Report(lab)

Correlator-Beamforming Delays and Fringe Test

Workspace

Recent Changes

Latest Aggregated Test Result (no tests)

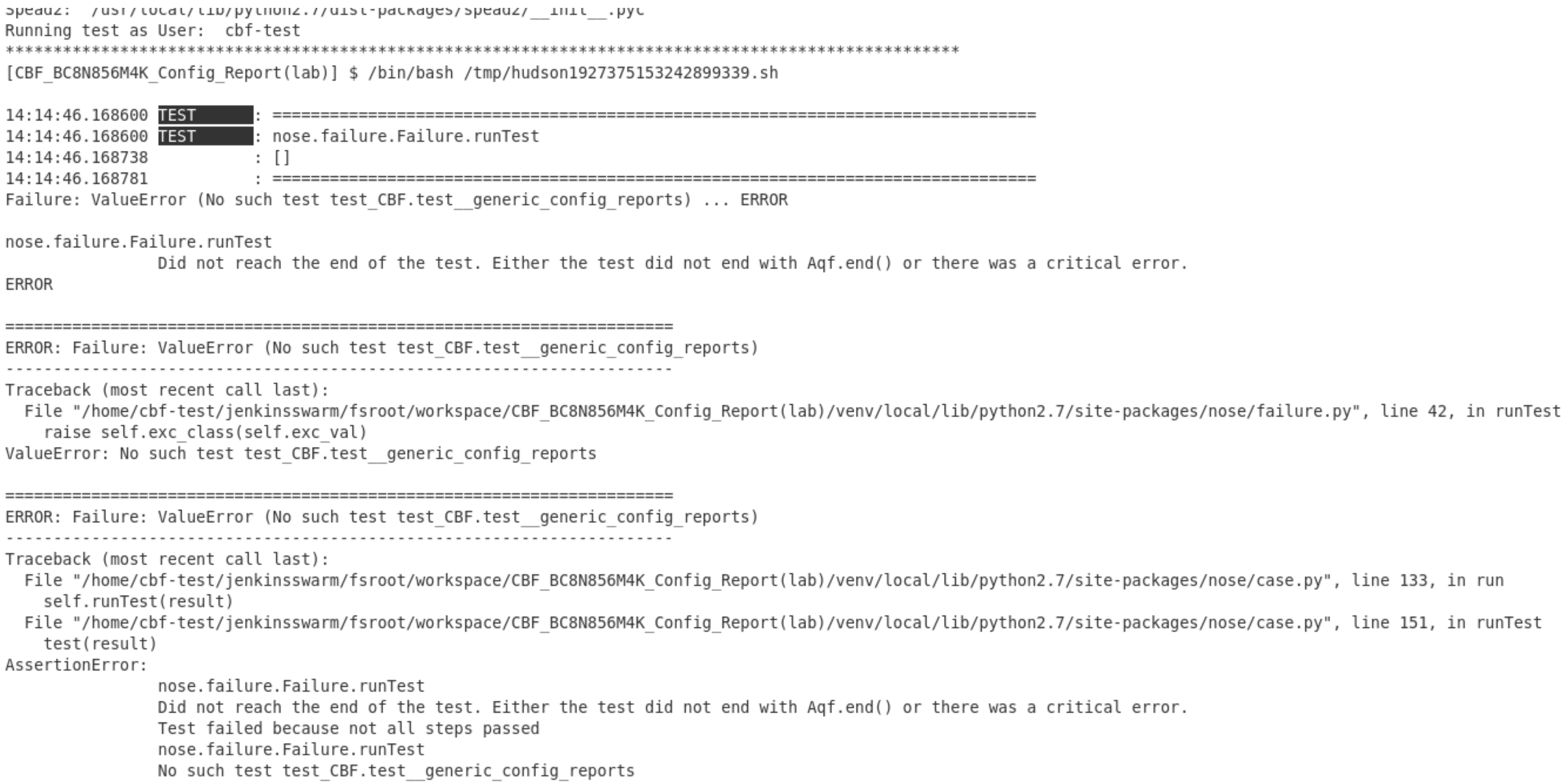
Latest Test Result (no tests)

Permalinks

- Last build (#3), 14 days ago
- Last stable build (#2), 14 days ago
- Last successful build (#2), 14 days ago
- Last failed build (#3), 14 days ago
- Last unsuccessful build (#3), 14 days ago
- Last completed build (#3), 14 days ago

Jenkins-Jira bug report API

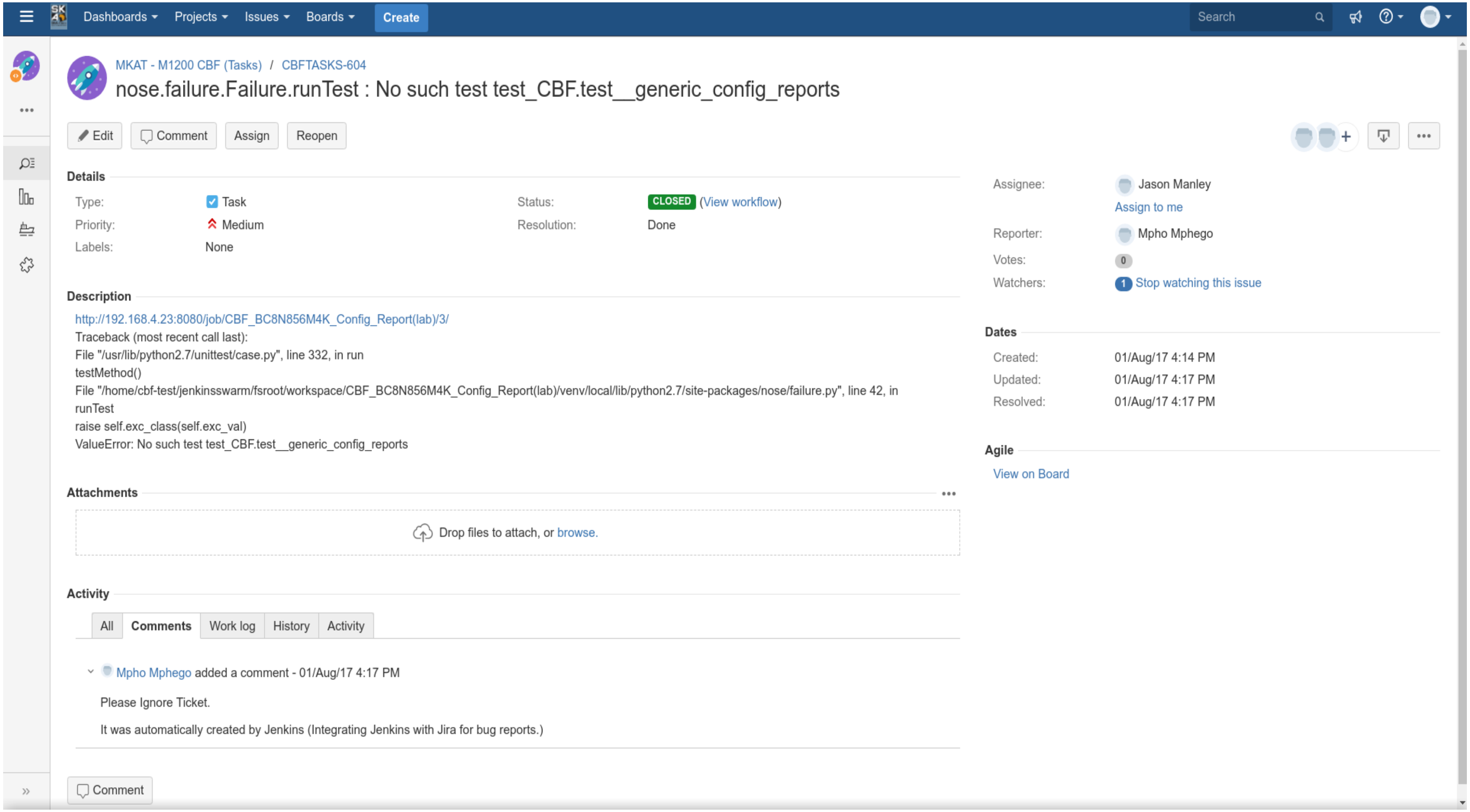
- Failed Test, executes Jira bug report API



```
spead2:  /usr/local/lib/python2.7/dist-packages/spead2/__init__.pyc
Running test as User:  cbf-test
****************************************************************************************************
[CBF_BC8N856M4K_Config_Report(lab)] $ /bin/bash /tmp/hudson1927375153242899339.sh

14:14:46.168600 TEST      : ================================================================================
14:14:46.168600 TEST      : nose.failure.Failure.runTest
14:14:46.168738           : []
14:14:46.168781           : ================================================================================
Failure: ValueError (No such test test_CBF.test__generic_config_reports) ... ERROR

nose.failure.Failure.runTest
                Did not reach the end of the test. Either the test did not end with Aqf.end() or there was a critical error.
ERROR

======================================================================
ERROR: Failure: ValueError (No such test test_CBF.test__generic_config_reports)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/cbf-test/jenkinsswarm/fsroot/workspace/CBF_BC8N856M4K_Config_Report(lab)/venv/local/lib/python2.7/site-packages/nose/failure.py", line 42, in runTest
    raise self.exc_class(self.exc_val)
ValueError: No such test test_CBF.test__generic_config_reports

======================================================================
ERROR: Failure: ValueError (No such test test_CBF.test__generic_config_reports)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/cbf-test/jenkinsswarm/fsroot/workspace/CBF_BC8N856M4K_Config_Report(lab)/venv/local/lib/python2.7/site-packages/nose/case.py", line 133, in run
    self.runTest(result)
  File "/home/cbf-test/jenkinsswarm/fsroot/workspace/CBF_BC8N856M4K_Config_Report(lab)/venv/local/lib/python2.7/site-packages/nose/case.py", line 151, in runTest
    test(result)
AssertionError:
                nose.failure.Failure.runTest
                Did not reach the end of the test. Either the test did not end with Aqf.end() or there was a critical error.
                Test failed because not all steps passed
                nose.failure.Failure.runTest
                No such test test_CBF.test__generic_config_reports
```

Jenkins-Jira bug report API (cont)

- Jira issues a bug report, for tracking.
  https://wiki.jenkins.io/display/JENKINS/JIRA+Plugin



MKAT - M1200 CBF (Tasks) / CBFTASKS-604

nose.failure.Failure.runTest : No such test test_CBF.test__generic_config_reports

Details

Type:	☑ Task	Status:	CLOSED (View workflow)
Priority:	Medium	Resolution:	Done
Labels:	None		

Description

```
http://192.168.4.23:8080/job/CBF_BC8N856M4K_Config_Report(lab)/3/
Traceback (most recent call last):
File "/usr/lib/python2.7/unittest/case.py", line 332, in run
testMethod()
File "/home/cbf-test/jenkinsswarm/fsroot/workspace/CBF_BC8N856M4K_Config_Report(lab)/venv/local/lib/python2.7/site-packages/nose/failure.py", line 42, in
runTest
raise self.exc_class(self.exc_val)
ValueError: No such test test_CBF.test__generic_config_reports
```

Attachments

Drop files to attach, or browse.

Activity

Mpho Mphego added a comment - 01/Aug/17 4:17 PM

Please Ignore Ticket.

It was automatically created by Jenkins (Integrating Jenkins with Jira for bug reports.)

Assignee:	Jason Manley Assign to me
Reporter:	Mpho Mphego
Votes:	0
Watchers:	1 Stop watching this issue

Dates

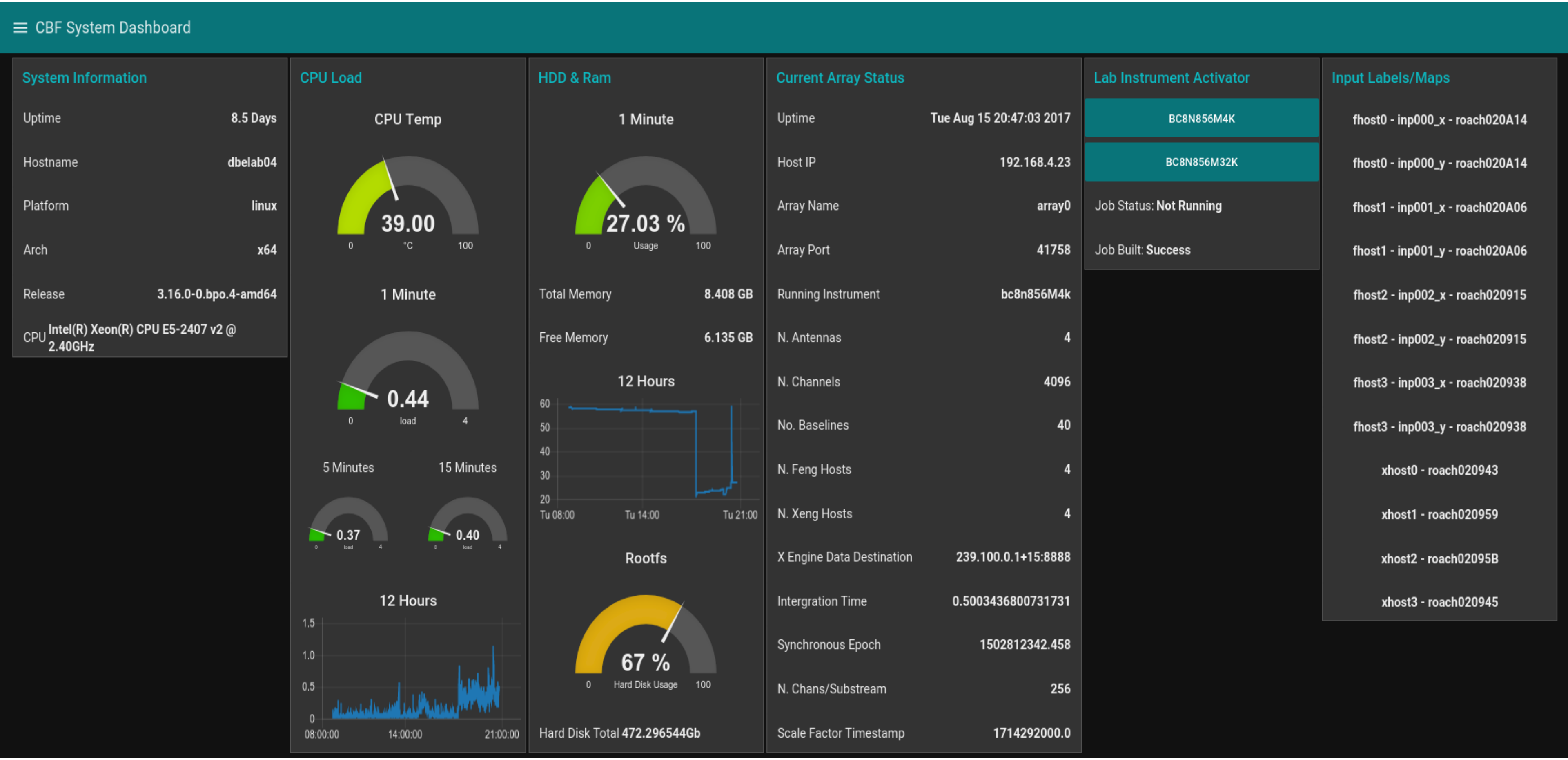
Created:	01/Aug/17 4:14 PM
Updated:	01/Aug/17 4:17 PM
Resolved:	01/Aug/17 4:17 PM

Agile

View on Board

CBF System Dashboard

- Docker based, Node-Red dashboard
- MQTT protocol is used to retrieve all relevant information, and then parses the data to dashboard.



≡ CBF System Dashboard

System Information

Uptime	8.5 Days
Hostname	dbelab04
Platform	linux
Arch	x64
Release	3.16.0-0.bpo.4-amd64
CPU	Intel(R) Xeon(R) CPU E5-2407 v2 @ 2.40GHz

CPU Load

CPU Temp

1 Minute

5 Minutes

15 Minutes

12 Hours

HDD & Ram

1 Minute

Total Memory	8.408 GB
Free Memory	6.135 GB

12 Hours

Rootfs

Hard Disk Total 472.296544Gb

Current Array Status

Uptime	Tue Aug 15 20:47:03 2017
Host IP	192.168.4.23
Array Name	array0
Array Port	41758
Running Instrument	bc8n856M4k
N. Antennas	4
N. Channels	4096
No. Baselines	40
N. Feng Hosts	4
N. Xeng Hosts	4
X Engine Data Destination	239.100.0.1+15:8888
Intergration Time	0.5003436800731731
Synchronous Epoch	1502812342.458
N. Chans/Substream	256
Scale Factor Timestamp	1714292000.0

Lab Instrument Activator

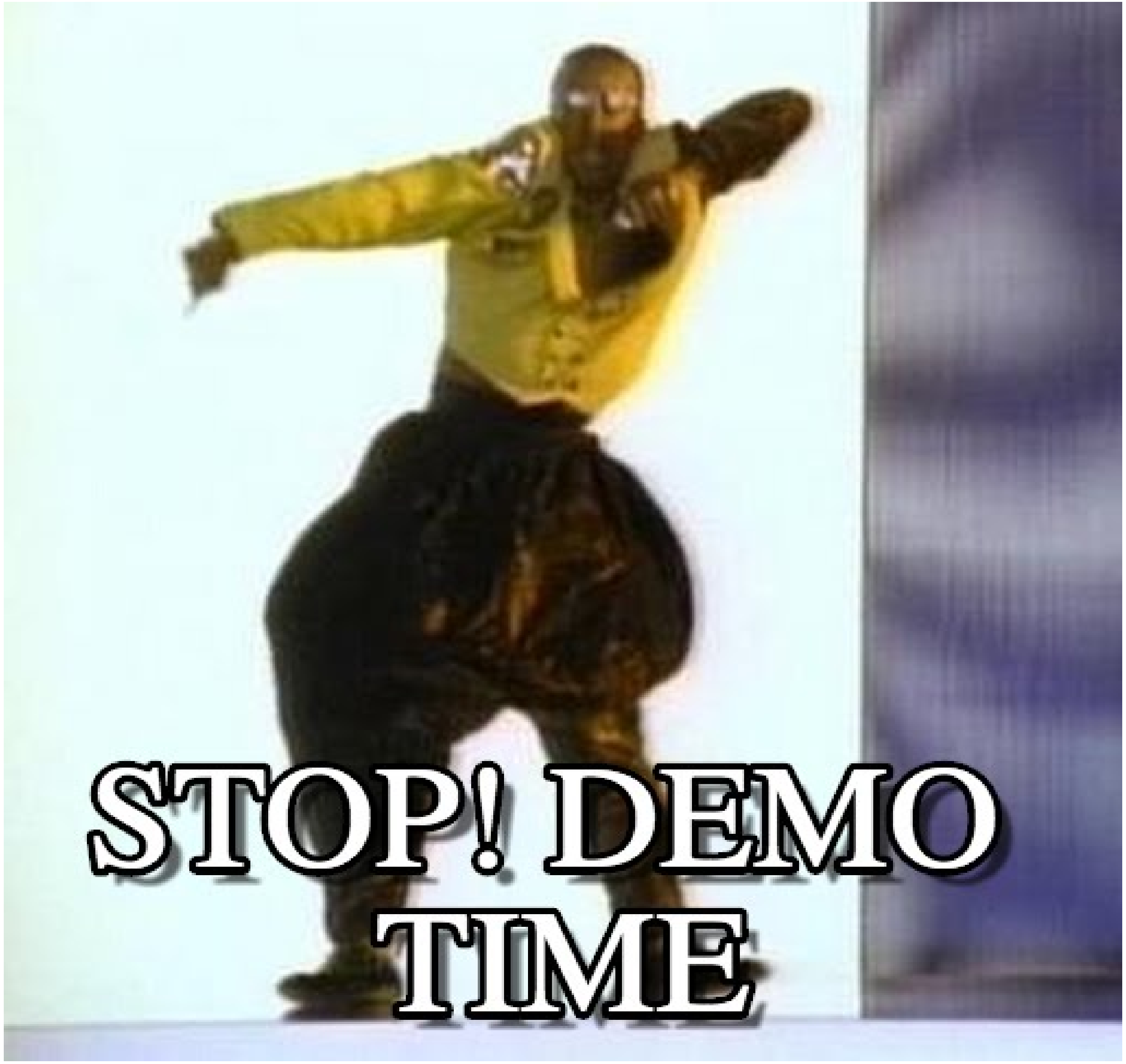
Job Status: **Not Running**

Job Built: **Success**

Input Labels/Maps

fhost0 - inp000_x - roach020A14
fhost0 - inp000_y - roach020A14
fhost1 - inp001_x - roach020A06
fhost1 - inp001_y - roach020A06
fhost2 - inp002_x - roach020915
fhost2 - inp002_y - roach020915
fhost3 - inp003_x - roach020938
fhost3 - inp003_y - roach020938
xhost0 - roach020943
xhost1 - roach020959
xhost2 - roach02095B
xhost3 - roach020945

https://github.com/ska-sa/CBF-System-Dashboard


STOP! DEMO TIME

SKA South Africa, a Business Unit of the National Research Foundation.

We are building the Square Kilometre Array radio telescope (SKA), located in South Africa and eight other African countries, with part in Australia. The SKA will be the largest radio telescope ever built and will produce science that changes our understanding of the universe

Contact information

Mpho Mphego
Test and Verification Engineer
Email: mmphego@ska.ac.za