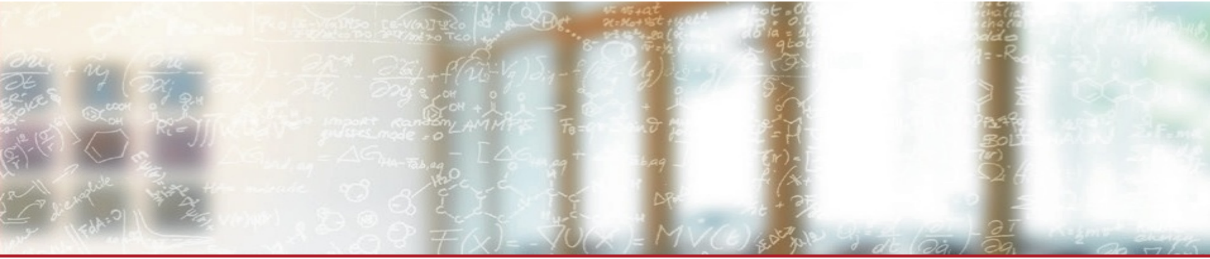
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Enabling Continuous Testing of HPC Systems using ReFrame

HPC System Testing: Procedures, Acceptance, Regression Testing, and Automation BoF
SC'19, Denver, CO, USA

Vasileios Karakasis, CSCS

November 21, 2019

reframe@cscs.ch
https://reframe-hpc.readthedocs.io
https://github.com/eth-cscs/reframe
https://reframe-slack.herokuapp.com

Why regression testing?

- The HPC software stack is highly complex and very sensitive to changes.
- How can we ensure that the user experience is unaffected after an upgrade or after an “innocent” change in the system configuration?
- How testing of such complex systems can be made sustainable?
  - Consistency
  - Maintainability
  - Automation

Background

- CSCS had a shell-script based regression testing suite
  - Tests very tightly coupled to system details
  - Lots of code replication across tests
  - 15K lines of test code and low coverage

- Simple changes required significant team effort

- Fixing even simple bugs was a tedious task

What is ReFrame?

An HPC testing framework that…

- allows writing **portable** HPC regression tests in Python,
- **abstracts away** the system interaction details,
- lets users focus solely on the **logic** of their test,
- provides a runtime for running **efficiently** the regression tests.

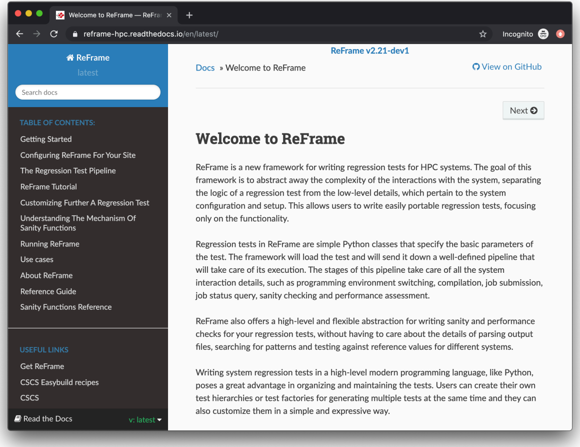


TABLE OF CONTENTS:

- Getting Started
- Configuring ReFrame For Your Site
- The Regression Test Pipeline
- ReFrame Tutorial
- Customizing Further A Regression Test
- Understanding The Mechanism Of Sanity Functions
- Running ReFrame
- Use cases
- About ReFrame
- Reference Guide
- Sanity Functions Reference

USEFUL LINKS

- Get ReFrame
- CSCS Easybuild recipes
- CSCS

Welcome to ReFrame

ReFrame is a new framework for writing regression tests for HPC systems. The goal of this framework is to abstract away the complexity of the interactions with the system, separating the logic of a regression test from the low-level details, which pertain to the system configuration and setup. This allows users to write easily portable regression tests, focusing only on the functionality.

Regression tests in ReFrame are simple Python classes that specify the basic parameters of the test. The framework will load the test and will send it down a well-defined pipeline that will take care of its execution. The stages of this pipeline take care of all the system interaction details, such as programming environment switching, compilation, job submission, job status query, sanity checking and performance assessment.

ReFrame also offers a high-level and flexible abstraction for writing sanity and performance checks for your regression tests, without having to care about the details of parsing output files, searching for patterns and testing against reference values for different systems.

Writing system regression tests in a high-level modern programming language, like Python, poses a great advantage in organizing and maintaining the tests. Users can create their own test hierarchies or test factories for generating multiple tests at the same time and they can also customize them in a simple and expressive way.

Key Features

- Support for cycling through programming environments and system partitions
- Support for different WLMs, parallel job launchers and modules systems
- Support for sanity and performance tests
- Support for test factories
- Support for container runtimes (new in v2.20)
- Support for test dependencies (new in v2.21)
- Concurrent execution of regression tests
- Progress and result reports
- Performance logging with support for Syslog and Graylog
- Clean internal APIs that allow the easy extension of the framework's functionality

Writing a Performance Test in ReFrame

```python
import reframe as rfm
import reframe.utility.sanity as sn


@rfm.simple_test
class Example7Test(rfm.RegressionTest):
    def __init__(self):
        self.descr = 'Matrix-vector␣multiplication␣(CUDA␣performance␣test)'
        self.valid_systems = ['daint:gpu']
        self.valid_prog_environs = ['PrgEnv-gnu', 'PrgEnv-cray', 'PrgEnv-pgi']
        self.sourcepath = 'example_matrix_vector_multiplication_cuda.cu'
        self.build_system = 'SingleSource'
        self.build_system.cxxflags = ['-O3']
        self.executable_opts = ['4096', '1000']
        self.modules = ['cudatoolkit']
        self.sanity_patterns = sn.assert_found(r'time␣for␣single␣matrix␣vector␣multiplication', self.stdout)
        self.perf_patterns = {
            'perf': sn.extractsingle(r'Performance:\s+(?P<Gflops>\S+)␣Gflop/s', self.stdout, 'Gflops', float)
        }
        self.reference = {
            'daint:gpu': {
                'perf': (50.0, -0.1, 0.1, 'Gflop/s'),
            }
        }
        self.tags = {'tutorial'}
```

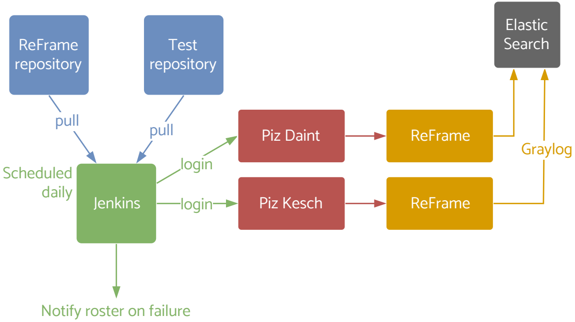
Running ReFrame

Sample failure

```
[==========] Running 1 check(s)
[==========] Started on Fri Jun  7 17:50:58 2019

[----------] started processing Example7Test (Matrix-vector multiplication using CUDA)
[ RUN      ] Example7Test on daint:gpu using PrgEnv-gnu
[     FAIL ] Example7Test on daint:gpu using PrgEnv-gnu
[----------] finished processing Example7Test (Matrix-vector multiplication using CUDA)

[  FAILED  ] Ran 1 test case(s) from 1 check(s) (1 failure(s))
[==========] Finished on Fri Jun  7 17:51:07 2019

===============================================================================
SUMMARY OF FAILURES
-------------------------------------------------------------------------------
FAILURE INFO for Example7Test
  * System partition: daint:gpu
  * Environment: PrgEnv-gnu
  * Stage directory: /path/to/stage/daint/gpu/PrgEnv-gnu/Example7Test
  * Job type: batch job (id=823427)
  * Maintainers: ['you-can-type-your-email-here']
  * Failing phase: performance
  * Reason: performance error: failed to meet reference: perf=50.358136, expected 70.0 (l=63.0, u=77.0)
-------------------------------------------------------------------------------
```

ReFrame @ CSCS

Tests and production setup

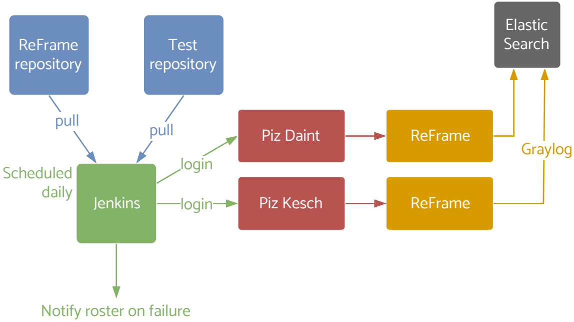


Several test categories identified by tags:

- Cray PE tests: only PE functionality
- Production tests: entire HPC software stack
- Maintenance tests: selection of tests for running before/after maintenance sessions
- Benchmarks
- 534 tests in total (most of them available on ReFrame's Github repo)

ReFrame @ CSCS

Tests and production setup



Several test categories identified by tags:

- Cray PE tests: only PE functionality
- Production tests: entire HPC software stack
- Maintenance tests: selection of tests for running before/after maintenance sessions
- Benchmarks
- 534 tests in total (most of them available on ReFrame's Github repo)

Experiences from Piz Daint's upgrade to CLE7:

- Enabling ReFrame as early as possible on the TDS has streamlined the upgrade process.
- Revealed several regressions in the programming environment that needed to be fixed.
- Builds confidence when finally everything is GREEN.

ReFrame @ CSCS

Test suite

- HPC applications: Amber, CP2K, CPMD, QuantumEspresso, GROMACS, LAMMPS, NAMD, OpenFoam, Paraview, TensorFlow
- Libraries: Boost, GridTools, HPX, HDF5, NetCDF, Magma, Scalapack, Trilinos, PETSc
- Programming environment: GPU, MPI, MPI+X functionality, OpenACC, CPU affinity
- Slurm functionality
- Performance and debugging tools
- I/O tests: IOR
- Microbenchmarks: CUDA, CPU, MPI
- Sarus container runtime checks
- OpenStack: S3 API

Conclusions

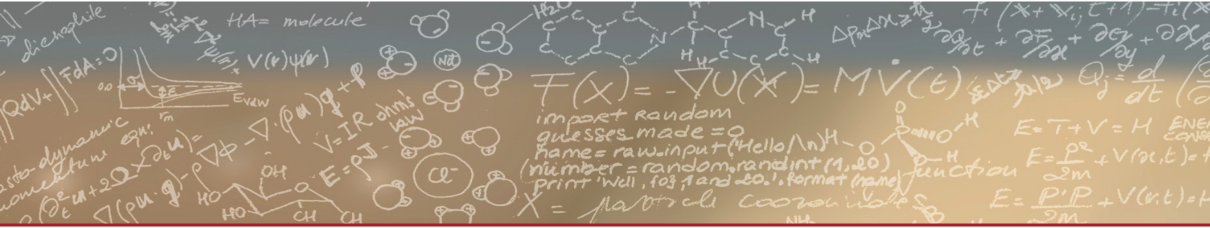
ReFrame is a powerful tool that allows you to continuously test an HPC environment without having to deal with the low-level system interaction details.

- High-level tests written in Python
- Portability across HPC system platforms
- Comprehensive reports and reproducible methods
- Easy integration in CI/CD workflows

Bug reports, feature requests, help @ https://github.com/eth-cscs/reframe

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich


HA= molecule
import random
guesses_made = 0
name = raw_input('Hello\n')
number = random.randint(1, 20)
print 'Well, {0}, I and 20.'.format(name)
E = T + V = H

Thank you for your attention

✉ reframe@cscs.ch

https://reframe-hpc.readthedocs.io

https://github.com/eth-cscs/reframe

https://reframe-slack.herokuapp.com