High Performance
Computing &
Big Data Services

hpc.uni.lu

hpc@uni.lu

@ULHPC

LUXEMBOURG
LET'S MAKE IT HAPPEN

Lmod

easybuild

RESIF 3.0: Toward a Flexible & Automated Management of User SW Environment on HPC facility

Dr. S. Varrette, Dr. E Kieffer, Dr. F. Pinel, Dr. E. Krishnasamy, S. Peter, H. Cartiaux and Dr. X. Besseron

University of Luxembourg (UL), Luxembourg

https://hpc.uni.lu

PEARC21

July 22th, 2021

Summary

University of Luxembourg

www.uni.lu

UNIVERSITÉ DU LUXEMBOURG

- **Created in 2003**, moved to Belval (South of the country) in 2015
- Among the top 250 universities in the Times Higher Education (THE) Rankings 2021
  - ↪ N°1 worldwide in the THE "international outlook" Rankings
  - ↪ N°20 worldwide in the THE Young University Rankings 2021.
    - ✓ N°4 (out of 64) in the THE Millennials Rankings 2021.

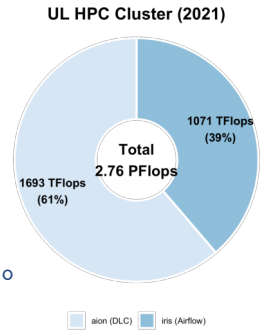
High Performance Computing @ UL

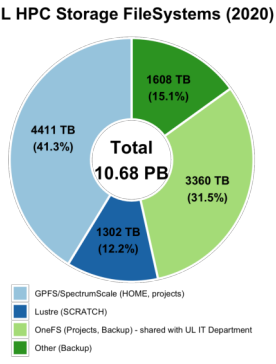
- **Started in 2007** (Dr. S. Varrette & Co.)
  - ↪ 2nd Largest HPC facility in Luxembourg after EuroHPC MeluXina

`hpc.uni.lu`

Technical Docs:
`hpc-docs.uni.lu`

ULHPC Tutorials:
`ulhpc-tutorials.rtfd.io`





High Performance Computing & Big Data Services

hpc.uni.lu
hpc@uni.lu
@ULHPC

HPC User Software Management: RESIF

- **Evolving** and **diverse** set of scientific apps built on **heterogeneous HPC** resources
  - ↪ notoriously complex, time-consuming, error-prone building process, **frequently repeated**
    - ✓ bug fix, support for new OS/MPI or dependencies (PMIx, UCX, Slurm or CUDA interfaces).
  - ↪ facilitated by novel software management frameworks Easybuild, Spack, Nix, GuixHPC...

HPC User Software Management: RESIF

- **Evolving** and **diverse** set of scientific apps built on **heterogeneous HPC** resources
  - ↪ notoriously complex, time-consuming, error-prone building process, **frequently repeated**
    - ✓ bug fix, support for new OS/MPI or dependencies (PMIx, UCX, Slurm or CUDA interfaces).
  - ↪ facilitated by novel software management frameworks Easybuild, Spack, Nix, GuixHPC...

- **RESIF**: **wrapper on top of Easybuild** meant to pilot user software generation
  - ↪ Fully **automates software set builds**, supports all avail. toolchains & *CPU/GPU* archs
    - ✓ incl. **automatic generation of environment modules/LMod**
  - ↪ Management of **software sets** for which different policies/roles sysadmin/[power]user
  - ↪ Clean (**hierarchical**) modules layout, **isolated arch. builds**, minimal requirements
  - ↪ Smooth **integration with streamline** development incl for contributing back
- **Reproducible** and **self-contained** deployment, with a **consistent** workflow
  - ↪ coupled with **strong versioning policy between environments**
  - ↪ automatic PR management for backward contribution to Easybuild community

Chronological developments

- **v1.0** (2014-2015) on Github: workflow definition, initial developments (S. Varrette, M. Schmitt)
  - ↪ Used to produce the following ULHPC software environments:
    - ✓ 2013-2015 software set (377 software packages)
    - ✓ 2015-2017 software set (133 software packages)

Chronological developments

- **v1.0** (2014-2015) on Github: workflow definition, initial developments (S. Varrette, M. Schmitt)
  - ↪ Used to produce the following ULHPC software environments:
    - ✓ 2013-2015 software set (377 software packages)
    - ✓ 2015-2017 software set (133 software packages)
- **v2.0** (2017-2019) code refactoring on private Gitlab instance (S. Peter, V. Plugaru)
  - ↪ Used to produce the following ULHPC software environments:
    - ✓ 2017-2018 software set (165 software packages)
    - ✓ 2018-2019 software set (210 software packages)
    - ✓ 2019-now software set (239 software packages)

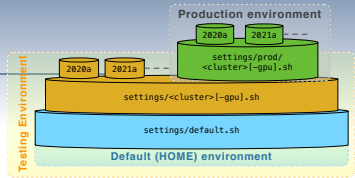
Chronological developments

- **v1.0** (2014-2015) on Github: workflow definition, initial developments (S. Varrette, M. Schmitt)
  - ↪ Used to produce the following ULHPC software environments:
    - ✓ 2013-2015 software set (377 software packages)
    - ✓ 2015-2017 software set (133 software packages)

- **v2.0** (2017-2019) code refactoring on private Gitlab instance (S. Peter, V. Plugaru)
  - ↪ Used to produce the following ULHPC software environments:
    - ✓ 2017-2018 software set (165 software packages)
    - ✓ 2018-2019 software set (210 software packages)
    - ✓ 2019-now software set (239 software packages)

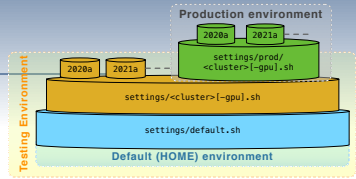
- YET, after 3 releases with RESIF 2:
  - ↪ complex workflow, 2 side repositories to maintain, no support for multiple architectures
  - ↪ explosion of custom easyconfigs (565!) and divergence from streamline Easybuild
    - ✓ no benefit from the worldwide EB HPC community
    - ✓ never any contribution back to streamline Easybuild

RESIF 3 Architecture-At-a-Glance



- **Single (control) repository** `ULHPC/sw`
  - ↪ hosted on private instance (Gitlab,gitolite), public export on Github `ULHPC/sw`
    - ✓ Git-LFS (Large File Storage) extension required for specialized software sources/install kits
    - ✓ `MkDocs`-based documentation [offline] rendering
  - ↪ defines roles/config settings organized by priority in a hierarchical way — Puppet Hiera inspired
  - ↪ custom easyconfigs (**ULHPC bundles**, MPI, commercial etc.) — restricted to the strict minimum
  - ↪ prevent divergence from streamline EB developments: customization outsourced by **hooks**
    - ✓ python callback functions that can be called during the different execution steps of Easybuilds.
    - ✓ `parse_hook` function — inject on the fly extras parameters in loaded easyconfig **before** their processing to simulate
  - ↪ launcher scripts (Slurm) used to deploy, complete or test the software stack
- Software/Modules organized according to `CategorizedModuleNamingScheme` — now part of EB
  - ↪ `<category>/<app>/<version>-<toolchain><versionsuffix>`
  - ↪ Lmod engine (with module usage tracking enabled)

RESIF 3 Architecture-At-a-Glance



- Global path for RESIF-related folder: `$RESIF_ROOT_DIR/{<cluster>|local}`
  - ↪ Flexible management of different deployment scenarios.

Operation Mode	`$RESIF_ROOT_DIR`	Activation source `settings/[...]`	Custom configs
Home builds	`<path/to/resif>/apps`	`default.sh`	
Testing builds (*shared* project)	`/work/projects/sw/resif`	`[<version>]/<cluster>[-gpu].sh`	`g+w`, setgid bit
Production `<version>` builds (`resif` user)	`/opt/apps/resif`	`prod/<version>/<cluster>[-gpu].sh`	

- **[bi-]Yearly** `<version>` release, aligned with EB toolchain releases
  - ↪ see foss & intel toolchains
  - ≃ 6 months of validation/import delay after EB release
- The EasyBuild software and modules installation (`$EASYBUILD_PREFIX`)
  - ↪ General format: `${RESIF_ROOT_DIR}/<cluster>/<version>/<arch>`
    - ✓ **Ex**: `iris` cluster: `${RESIF_ROOT_DIR}/iris/<version>/<broadwell|skylake|gpu>`
    - ✓ **Ex**: `aion` cluster: `${RESIF_ROOT_DIR}/aion/<version>/epyc`

RESIF 3 Architecture-At-a-Glance

- Global path for **easyconfigs** (`$EASYBUILD_ROBOT_PATHS`)
  - ↪ `make fork-easyconfigs-update` to update your local `easybuild-easyconfigs` **Fork**
  - ↪ transparent integration of streamline developments
    - ✓ used also for automatic PR management through EB Github CLI integration
    - ✓ requires `keyring` python package (or alternative) to store Personal GitHub token

1. `<path/to/resif>/easyconfigs`	Custom ULHPC easyconfigs
2. `${DEFAULT_ROBOT_PATHS}`	Default path from **loaded** `tools/EasyBuild` module
3. `<path/to/fork>/easybuild/easyconfigs`	[ULHPC] Fork copy of official `easybuilders/easyconfigs` repo

RESIF 3 Architecture-At-a-Glance

- Global path for **easyconfigs** (`$EASYBUILD_ROBOT_PATHS`)
  - ↪ `make fork-easyconfigs-update` to update your local `easybuild-easyconfigs` **Fork**
  - ↪ transparent integration of streamline developments
    - ✓ used also for automatic PR management through EB Github CLI integration
    - ✓ requires `keyring` python package (or alternative) to store Personal GitHub token

1. `<path/to/resif>/easyconfigs`	Custom ULHPC easyconfigs
2. `${DEFAULT_ROBOT_PATHS}`	Default path from **loaded** `tools/EasyBuild` module
3. `<path/to/fork>/easybuild/easyconfigs`	[ULHPC] Fork copy of official `easybuilders/easyconfigs` repo

- Global path for install kits / source files (`$EASYBUILD_SOURCEPATH`)

1. `${[LOCAL_]RESIF_ROOT_DIR}/sources`	'temporary' **common** sources for all clusters/arch
2. `${EASYBUILD_PREFIX}/sources`	specialized sources for the cluster/arch
3. `<path/to/resif>/sources`	**GIT-LFS** sources as part of the repository
4. `/opt/apps/sources`	previously downloaded sources common to all `<cluster>`

ULHPC Software Sets in RESIF 3

- **User Software Sets** now defined as native Easybuild Module `Bundle` easyblock
  ↪ **ULHPC** bundles, associated to toolchain version – see `easyconfigs/u/ULHPC*`

Bundle Name	Description	Featured applications
`ULHPC-<version>`	Default global bundle for 'regular' nodes	`ULHPC-*-<version>` (root bundle)
`ULHPC-toolchains-<version>`	Toolchains, compilers, debuggers, programming languages, MPI suits, Development tools and libraries	`GCCcore`, `foss`, `intel`, `LLVM`, `OpenMPI`, `CMake`, `Go`, `Java`, `Julia`, `Python`, `Spack`...
`ULHPC-bd-<version>`	Big Data	Apache `Spark`, `Flink`, `Hadoop`...
`ULHPC-bio-<version>`	Bioinformatics, biology and biomedical	`GROMACS`, `Bowtie2`, `TopHat`, `Trinity`...
`ULHPC-cs-<version>`	Computational science, incl. CAE, CFD, Chemistry, Earth Sciences, Physics and Materials Science	`ANSYS`, `OpenFOAM`, `ABAQUS`, `NAMD`, `GDAL`, `QuantumExpresso`, `VASP`...
`ULHPC-dl-<version>`	AI / Deep Learning / Machine Learning	`TensorFlow`, `PyTorch`, `Horovod`...
`ULHPC-math-<version>`	High-level mathematical software and Optimizers	`R`, `MATLAB`, `CPLEX`, `GEOS`, `GMP`, `Gurobi`...
`ULHPC-perf-<version>`	Performance evaluation / Benchmarks	`ArmForge`, `PAPI`, `HPL`, `IOR`, `Graph500`...
`ULHPC-tools-<version>`	General purpose tools	`DMTC`, `Singularity`, `gocryptfs`...
`ULHPC-visu-<version>`	Visualization, plotting, documentation & typesetting	`OpenCV`, `ParaView`...
`ULHPC-gpu-<version>`	Specific GPU/CUDA-accelerated software	`{foss,intel}cuda`, `cuDNN`, `TensorFlow`, `PyTorch`, `GROMACS`...

ULHPC Software Sets in RESIF 3

```
easyblock = "Bundle"
name = 'ULHPC-toolchains'
version = '2019b'
homepage = 'http://hpc.uni.lu/'
description = """[...]"""
toolchain = SYSTEM
local_gccver      = '8.3.0'
local_intelver    = '2019.5.281'
local_binutilsver = '2.32'
local_pyver       = '3.7.4'
local_llvmver     = '9.0.1'
dependencies = [
    #-----------------------------
    # Main toolchains / Compilers
    #-----------------------------
    ('GCCcore', local_gccver),
    ('foss',    version),
    ('intel',   version),
    # Other useful compiling framworks
    ('LLVM',   local_llvmver, '', ('GCCcore', local_gccver)),
    ('Clang',  local_llvmver, '', ('GCCcore', local_gccver)),
    ('PGI',    '19.10',     '-%s'%(local_binutilsver), ('GCC', local_gccver)),
    [...]
```

Toolchain Component	Software set release <version> 2019a (*deprecated*)	2019b old	2020a prod	2021a* devel
GCCCore	8.2.0	8.3.0	9.3.0	10.3.0
foss	2019a	2019b	2020a	2021a
intel	2019a	2019b	2020a	2021a
binutils	2.31.1	2.32	2.34	2.36
Python	3.7.2 (2.7.15)	3.7.4 (2.7.16)	3.8.2 (2.7.18)	3.9.2
LLVM	8.0.0	9.0.1	10.0.1	11.1.0
OpenMPI	3.1.4	3.1.4	4.0.3	4.1.1
RESIF version	2.0 (*old*)	3.0	3.0	3.1
#Modules:	229	<arch>: 269 gpu: 135	<arch>: 274 gpu: 151	<arch>: n/a gpu: n/a

ULHPC Software Set Build Jobs

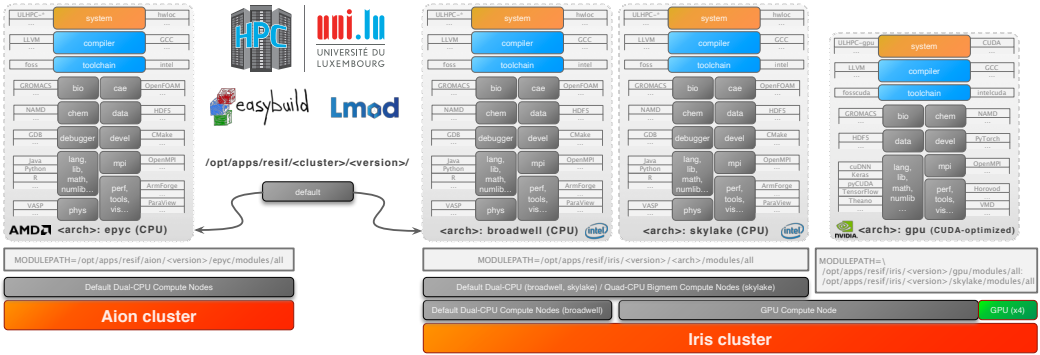
- **Slurm launchers** to submit/build ULHPC bundles under `scripts/` Build time Ex: 184h 2019b skylake
  - ↪ WIP: dominance tree representation of bundle SW dependency to decrease builds time
    - ✓ quickly identify intermediate dominating software which can be built within concurrent jobs

Operation Mode	Architecture	Launcher script
Easybuild bootstrap/update	*	`setup.sh`
Home/**Testing** builds	default	`[sbatch] ./scripts/[<version>]/launcher-test-build-cpu.sh`
	CPU non-default	`[sbatch] ./scripts/[<version>]/launcher-test-build-cpu-<arch>.sh`
	GPU optimized	`[sbatch] ./scripts/[<version>]/launcher-test-build-gpu.sh`
Production <version> builds	default	`[sbatch] ./scripts/prod/launcher-resif-prod-build-cpu.sh -v <version>`
	CPU non-default	`[sbatch] ./scripts/prod/launcher-resif-prod-build-cpu-<arch>.sh -v <version>`
	GPU optimized	`[sbatch] ./scripts/prod/launcher-resif-prod-build-gpu.sh -v <version>`

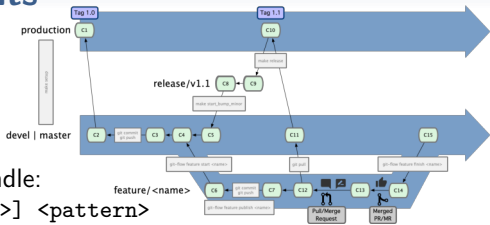
```
$ ./scripts/launcher-test-build-cpu.sh -h
NAME
  launcher-test-build-cpu.sh: RESIF 3.0 launcher for ULHPC User Software building
          Based on UL HPC Module Bundles ULHPC-[<topic>-]<version>.eb
          Default CPU Builds compliant with all CPU archictectures in place
USAGE
  [sbatch] ./scripts/launcher-resif-build-cpu.sh [-n] [-D] [ toolchains | bio | cs | dl | gpu | math | perf | tools | visu ]
  [sbatch] ./scripts/launcher-resif-build-cpu.sh [-n] [-D] path/to/file.eb
```

ULHPC Software Modules Organization

- `MODULEPATH` adapted for each architecture to take the best out of the optimized builds

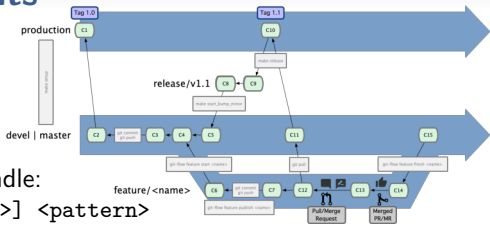


ULHPC Bundle Developments



- Hybrid Git-flow/Github-Flow **git workflow**
  - ↪ Pre-requisite: repo+streamline easyconfigs updates
    `make up; make fork-easyconfigs-update`
- Finding a software to integrate into current/new bundle:
  - ↪ `./scripts/suggest-easyconfigs [-v <version>] <pattern>`
    1. **new version found** for the target release: update the bundle
    2. **old/obsolete version found**: adapt and provide a new easyconfig (+PR), update the bundle
    3. **none found**: create a new easyconfig (+PR), update the bundle

ULHPC Bundle Developments



- Hybrid Git-flow/Github-Flow **git workflow**
  - ↪ Pre-requisite: repo+streamline easyconfigs updates `make up; make fork-easyconfigs-update`
- Finding a software to integrate into current/new bundle:
  - ↪ `./scripts/suggest-easyconfigs [-v <version>] <pattern>`
    1. **new version found** for the target release: update the bundle
    2. **old/obsolete version found**: adapt and provide a new easyconfig (+PR), update the bundle
    3. **none found**: create a new easyconfig (+PR), update the bundle
- Licenses and Keys managed by hooks to avoid diverging from streamline easyconfigs
  - ↪ `hooks/ulhpc.py`
- Restrict at max. the **internal** custom Easyconfigs by **contributing back to Easybuild**
  - ↪ remove designed easyconfigs once PR merged in official streamline developments

Contributions to streamline Developments

- Assumes proper Github integration (personnal token, keyring etc.)
  ↪ Set of helper scripts provided in RESIF to automate the process in a consistent workflow

```
# Creating a new pull requests (typically ion your laptop)
./scripts/PR-create -n easyconfigs/<letter>/<software>/<filename>.eb    # Dry-run
./scripts/PR-create easyconfigs/<letter>/<software>/<filename>.eb
# Complete it with a successfull test report ON IRIS/AION
sbatch ./scripts/PR-rebuild-upload-test-report.sh <ID>

# (eventually) Update/complete the pull-request with new version/additional EB files
eb --update-pr <ID> <file>.eb --pr-commit-msg "<message>" # use native easybuild command here
#  Update your local easyconfigs from remote PR commits
./scripts/update-from-PR [-n] <ID>

# Repo cleanup upon merged pull-request
./scripts/PR-close [-n] <ID>
```

Creating new Pull-Requests *[docs/contributing/pull-requests.md]*

```
./scripts/PR-create [-n]  easyconfigs/<letter>/<software>/<filename>.eb
```

- aka submitting working Easyconfigs to the official streamline repository for integration.
  1. checking code style with `eb --check-contrib <ebfile>`
  2. submitting a new pull requests using `eb --new-pr <ebfile>`
  3. store info on pending PR in a dedicated directory `easyconfigs/pull-requests/<ID>`
     - ✓ create the directory `easyconfigs/pull-requests/<ID>`
     - ✓ add symlink to the EB file
     - ✓ add and commit all files, including the EB file
- Now you want to **upload a test report**, **from `<cluster>`**, using the script:

```
./scripts/PR-rebuild-upload-test-report.sh <ID>
```

Handle Pull-Request Life Cycle *[docs/contributing/pull-requests.md]*

- **Update/complete the pull-request** with your corrections / add new EB files.
  ↪ `eb --update-pr <ID> <file>.eb --pr-commit-msg "<message>"`
- **Update your local easyconfigs from PR commits**
  ↪ commits might be proposed by easybuilders to correct your initial easyconfig

```
./scripts/update-from-PR [-n] <ID>
```

1. Collect info on the Pull request using the Github API
   ↪ resulting JSON stored under `logs/$(date +%F)-pull-request-<ID>.json`
2. synchronize your local copy of the (fork) easyconfigs repository
3. checkout the PR branch, get latest commits/updates in the (fork) easyconfigs repository
4. update local easyconfigs files accordingly, checkout back to `develop` in the fork repo
5. delete the JSON file holding the REST API request

Closing Merged Pull Requests *[docs/contributing/closing-merged-pr.md]*

```
./scripts/PR-close [-n] <ID>
```

- **Once a PR is accepted** & merged, time to make some **cleanup in the RESIF repo**!
  - ↪ this custom easyconfigs are now integrated in streamline repo and no longer required
  1. Collect info on the Pull request using the Github API
  2. delete from git the directory `easyconfigs/pull-requests/<ID>`
     - ✓ the symlinks under `easyconfigs/pull-requests/<ID>/*.eb`
     - ✓ the target real files `easyconfigs/<letter>/<software>/<filename>.eb`
  3. synchronize your local copy of the (fork) easyconfigs repository
  4. delete the git branch(es) (including remotes) used for the pull request
  5. delete the JSON file holding the REST API request

Conclusion

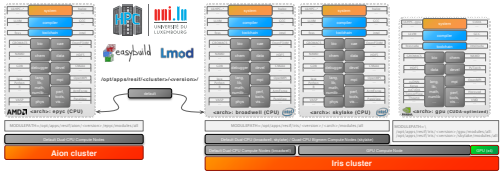
- **RESIF 3 validated successfully** on **two production HPC systems**
  - ↪ Iris supercomputer: Intel `broadwell`/`skylake` CPUs, NVidia V100 GPUs
  - ↪ Aion supercomputer: AMD `epyc`, no accelerators

Cluster	Date	Vendor	Proc. Description	#N	#C	R_{peak}
aion	2021	Atos	AMD EPYC 7H12 @2.6 GHz 2 × 64c, 256GB	318	40704	1693,29 TFlops
			`aion` **TOTAL:**	**318**	**40704**	**1693.3 TFlops**
iris	2017	Dell	Intel Xeon E5-2680 v4@2.4GHz 2 × 14C,128GB	108	3024	116,12 TFlops
	2018	Dell	Intel Xeon Gold 6132 @ 2.6 GHz 2 × 14C,128GB	60	1680	139,78 TFlops
	2018	Dell	Intel Xeon Gold 6132 @ 2.6 GHz 2 × 14C,768GB	24	672	55,91 TFlops
	2019		*Per node:* 4x NVIDIA Tesla V100 SXM2 16/32GB	96 GPUs	491520	748,8 GPU TFlops
	2018	Dell	Intel Xeon Platinum 8180M @ 2.5 GHz 4 × 28C,3072GB	4	448	35,84 TFlops
			`iris` **TOTAL:**	**196**	**5824**	**347.65 TFlops**
				96 GPUs	491520	**+748.8 GPU Tflops**

Conclusion

- **RESIF 3 validated successfully** on **two production HPC systems**
  - ↪ Iris supercomputer: Intel `broadwell`/`skylake` CPUs, NVidia V100 GPUs
  - ↪ Aion supercomputer: AMD `epyc`, no accelerators
- In production since May 2020, successful collaborative team workflow
  - ↪ **optimized builds** organized by architecture, exposed through Lmod
    - ✓ $\simeq$ 90% reduction of required custom easyconfigs compared to RESIF2
    - ✓ **Latest release**: [3×]274 software packages + 151 GPU-optimized
  - ↪ Smooth **integration with streamline** development – codebase on Github
- **WIP**: ReFrame binding, coupled with build/validation time optimization

Toolchain / Component	Software set release <version> 2019a (*deprecated*)	2019b old	2020a prod	2021a* devel
GCCcore	8.2.0	8.3.0	9.3.0	10.3.0
foss	2019a	2019b	2020a	2021a
intel	2019a	2019b	2020a	2021a
binutils	2.31.1	2.32	2.34	2.36
Python	3.7.2 (2.7.15)	3.7.4 (2.7.16)	3.8.2 (2.7.18)	3.9.2
LLVM	8.0.0	9.0.1	10.0.1	11.1.0
OpenMPI	3.1.4	3.1.4	4.0.3	4.1.1
RESIF version	2.0 (*old*)	3.0	3.0	3.1
#Modules:	229	<arch>: 269 gpu: 135	<arch>: 274 gpu: 151	<arch>: n/a gpu: n/a



Thank you for your attention...

Questions?

Sebastien Varrette, Emmanuel Kieffer, Frederic Pinel, Ezhilmathi Krishnasamy, Sarah Peter, Hyacinthe Cartiaux and Xavier Besseron

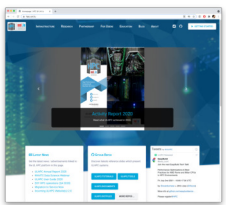
RESIF 3.0: Toward a Flexible & Automated Management of User Software Environment on HPC facility ACM PEARC'21

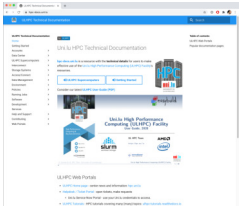
University of Luxembourg, Belval Campus:
Maison du Nombre, 4th floor
2, avenue de l'Université
L-4365 Esch-sur-Alzette
mail: firsname.lastename@uni.lu

High Performance Computing @ Uni.lu
mail: hpc@uni.lu

High Performance Computing @ Uni.lu

www: `hpc[-docs].uni.lu`





RESIF 3 Codebase: `ULHPC/sw`