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

Sebastien Varrette
University of Luxembourg
Esch-sur-Alzette, Luxembourg
sebastien.varrette@uni.lu

Emmanuel Kieffer
University of Luxembourg
Esch-sur-Alzette, Luxembourg
Emmanuel.Kieffer@uni.lu

Frederic Pinel
University of Luxembourg
Esch-sur-Alzette, Luxembourg
Frederic.Pinel@uni.lu

Ezhilmathi Krishnasamy
University of Luxembourg
Esch-sur-Alzette, Luxembourg
ezhilmathi.krishnasamy@uni.lu

Sarah Peter
University of Luxembourg
Esch-sur-Alzette, Luxembourg
sarah.peter@uni.lu

Hyacinthe Cartiaux
University of Luxembourg
Esch-sur-Alzette, Luxembourg
hyacinthe.cartiaux@uni.lu

Xavier Besseron
University of Luxembourg
Esch-sur-Alzette, Luxembourg
Xavier.Besseron@uni.lu

ABSTRACT

High Performance Computing (HPC) is increasingly identified as a strategic asset and enabler to accelerate the research and the business performed in all areas requiring intensive computing and large-scale Big Data analytic capabilities. The efficient exploitation of heterogeneous computing resources featuring different processor architectures and generations, coupled with the eventual presence of GPU accelerators, remains a challenge. The University of Luxembourg operates since 2007 a large academic HPC facility which remains one of the reference implementation within the country and offers a cutting-edge research infrastructure to Luxembourg public research. The HPC support team invests a significant amount of time (*i.e.*, several months of effort per year) in providing a software environment optimised for hundreds of users, but the complexity of HPC software was quickly outpacing the capabilities of classical software management tools. Since 2014, our scientific software stack is generated and deployed in an automated and consistent way through the RESIF framework, a wrapper on top of Easybuild and Lmod [5] meant to efficiently handle user software generation. A large code refactoring was performed in 2017 to better handle different software sets and roles across multiple clusters, all piloted through a dedicated control repository. With the advent in 2020 of a new supercomputer featuring a different CPU architecture, and to mitigate the identified limitations of the existing framework, we report in this state-of-practice article RESIF 3.0, the latest iteration of our scientific software management suit now relying on streamline Easybuild. It permitted to reduce by around 90% the number of custom configurations previously enforced by specific Slurm and MPI settings, while sustaining optimised builds coexisting for different dimensions of CPU and GPU architectures. The workflow for contributing back to the Easybuild community was also automated and a current work in progress aims at drastically decrease the building time of a complete software set generation. Overall, most design choices for our wrapper have been motivated by several years of experience in addressing in a flexible and convenient way the heterogeneous needs inherent to an academic environment aiming for research excellence. As the code base is available publicly, and as we wish to transparently report also the pitfalls and difficulties met, this tool may thus help other HPC centres to consolidate their own software management stack.

CCS CONCEPTS

• **Software and its engineering** → **Software creation and management**; **Software infrastructure**; • **Computer systems organization** → *Parallel architectures*; • **Applied computing** → *Computer-aided design.*

ACM Reference Format:
Sebastien Varrette, Emmanuel Kieffer, Frederic Pinel, Ezhilmathi Krishnasamy, Sarah Peter, Hyacinthe Cartiaux, and Xavier Besseron. 2021. RESIF 3.0: Toward a Flexible & Automated Management of User Software Environment on HPC facility . In *Practice and Experience in Advanced Research Computing (PEARC '21), July 18–22, 2021, Boston, MA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3437359.3465600

1 INTRODUCTION

The University of Luxembourg (UL) operates since 2007 a large research computing facility which remains one of the reference implementation within the country [10]. The ULHPC centre currently supports a wide user base ranging from University staff and students to research partners and commercial users. This includes computer scientists, engineers, physicists, material science researchers, biologists, economists and even historians, psychologists and social science researchers which are given the possibility to run compute- and storage-intensive computations as part of their research or training. The ULHPC facility currently offers two different clusters featuring heterogeneous CPU architectures (Intel and AMD) of different generations and 3 different types of nodes.

*Produces the permission block, and copyright information

PEARC '21, July 18–22, 2021, Boston, MA, USA

ACM ISBN 978-1-4503-8292-2/21/07.
https://doi.org/10.1145/3437359.3465600

To efficiently address the diverse applicative area needs, the ULHPC team frequently builds, installs, and supports a wide variety of scientific applications. Frequently, these applications must be rebuilt to fix bugs and offer updated versions of these software, or simply to support new versions of the Operating System (OS), MPI implementation, compiler, and other dependencies (for instance in case of updated PMIx [2], Slurm or CUDA interfaces). This building process, especially when it comes to tailor the software performance with heterogeneous computing resources, is notoriously complex. Luckily, novel software management frameworks such as EasyBuild [5] or Spack [4] have arisen and permitted to simplify the building process combined with the automatic generation of Environment Modules (compliant with LMod [8] in our case).

Since 2014, our scientific software stack is generated and deployed in an automated and consistent way through the RESIF framework (*Revolutionary EB-based Software Installation Framework*), a wrapper on top of Easybuild meant to pilot user software generation. The main objectives of this project was to fully automate software builds and to supports all available toolchains and *software sets* through a clean hierarchical modules layout to facilitate its usage and provide an intuitive interface to the users. We favoured an intermediate layout (*i.e.*, compared to the full hierarchical view) where the software modules are prefixed by a category level, taken out from one of the supported software category or *module class*, thus leading to the following directory layout: `<category>/<app>/<version>-<toolchain><versionsuffix>`. To that end, the `CategorizedModuleNamingScheme` was proposed and is now integrated as part of the available module naming schemes of Easybuild. Furthermore, at the heart of RESIF remains the management of *software sets* for which different policies (configuration sources etc.) and building roles (system administrator, end user etc.) are supported. We also wanted to facilitate the *reproducible* and *self-contained* deployment of the complete software stack, coupled with a strong versioning policy between environments and (typically) yearly release cycles. A large code review was performed in 2017 with the arrival of the Iris cluster to bring a YAML-based [1] description of the software sets, piloted through a dedicated control repository hosted on a private Gitlab instance. It also defines the roles and configuration sources organized by priority in a hierarchical way inspired from Puppet Hiera [9]. A second repository was setup to host custom *easyconfigs* (*i.e.,* Easybuild recipes for building software) adapted to the ULHPC platform characteristics (Slurm and MPI bindings, licences server details, etc.). While this second release sustained the deployment of 3 generations of software sets until 2019, the workflow proved to be quite complex and hard to maintain. Furthermore, the broken compliance with streamline EasyBuild developments led to an explosion of custom configurations. With the advent of a new supercomputer (named `Aion`) featuring a different CPU architecture (AMD Epyc instead of Intel Broadwell/Skylake), and to mitigate the identified limitations, a complete code refactoring was initiated leading to the RESIF 3.0 framework presented in this article. It shall bring the following benefits within an HPC or research computing center using it depending on the profile of the person interacting with this framework: *site managers* will see a more consistent and distributed workflow within the building operational team, improving the turn-around time as well as the software ecosystem quality; *system administrators* will optimized

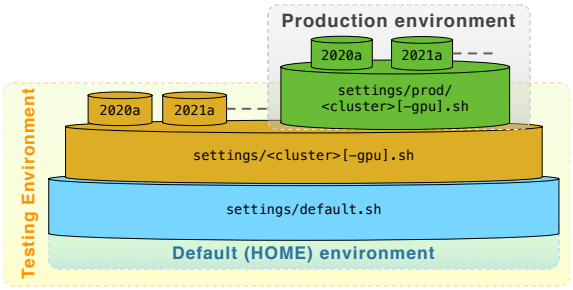

Figure 1: RESIF 3.0 settings hierarchy organization

the time spent on the user software stack management in a robust and systematic workflow tailored to meet the evolving user needs and expectations with minimal efforts. Finally, *end users* will be offered the possibility to easily extend the proposed software set and contribute to the environment developments, typically when it comes to test and bring to production an updated version of a given application.

2 RESIF 3.0 ARCHITECTURE-AT-A-GLANCE

RESIF developments are piloted from a single repository meant to be hosted on a private Gitlab or Gitolite instance (yet publicly available on Github[1]). It holds the configurations and setup tools, the launcher scripts used to deploy, complete or test the software stack, specialized software sources stored using the Git-LFS (Large File Storage) extension to complete the global directory hosting installation kits and source files, custom easyconfigs (of prime importance the *ULHPC bundles* defining the software sets to be built for a given release), documentation files managed by the `MkDocs`[7] suit allowing for both a local *offline* rendering and a remote deployment, and finally various scripts piloting the interaction with the Easybuild community developments discussed later.

The global path for RESIF-related generated files is identified by the environment variable `$RESIF_ROOT_DIR`. This permits to support three operation modes: (1) **home** builds for end-users, (2) **testing** builds for a *group* of users within a *shared* project directory – `/work/projects/sw` in the sequel and (3) **production**-ready builds operated by a dedicated user `resif` having write rights within the target `/opt/sys/apps` system directory.

To reflect these roles, configuration settings are embedded in the specific files provided under the `settings/` directory. Similarly to Puppet Hiera or Ansible variable structures, these settings are organized in a hierarchical way depicted in *Figure 1*. On top of this hierarchy, the `<version>` settings match the target release of the ULHPC software set which is aligned with the EasyBuild `foss` and `intel` toolchains release (Ex: `2020a` – see *Table 3*). Below the version layer stands the architecture-dependent settings aggregated within cluster settings, where the CPU architecture is automatically guessed. For instance, the Intel Skylake micro-architecture is detected from the availability of the AVX512 extension. For convenience, GPU settings are explicitly separated. Finally, all settings inherit from the *default* configuration used in particular for home-based deployments. All three deployment scenarios bring specialized Easybuild configurations summarized in the *Table 1*.

[1] See https://github.com/ULHPC/sw

Table 1: Deployment scenarios and associated configurations supported by RESIF 3.0

Operation Mode	$RESIF_ROOT_DIR	Activation (source settings/[...])	Custom Configurations
Home builds	$HOME/.easybuild/local	default.sh	
Testing builds (*shared* project)	/work/projects/sw/resif	[<version>/]<cluster>[-gpu].sh	Group write permissions & group ID bit
Production builds (resif user)	/opt/apps/resif	prod/[<version>/]<cluster>[-gpu].sh	

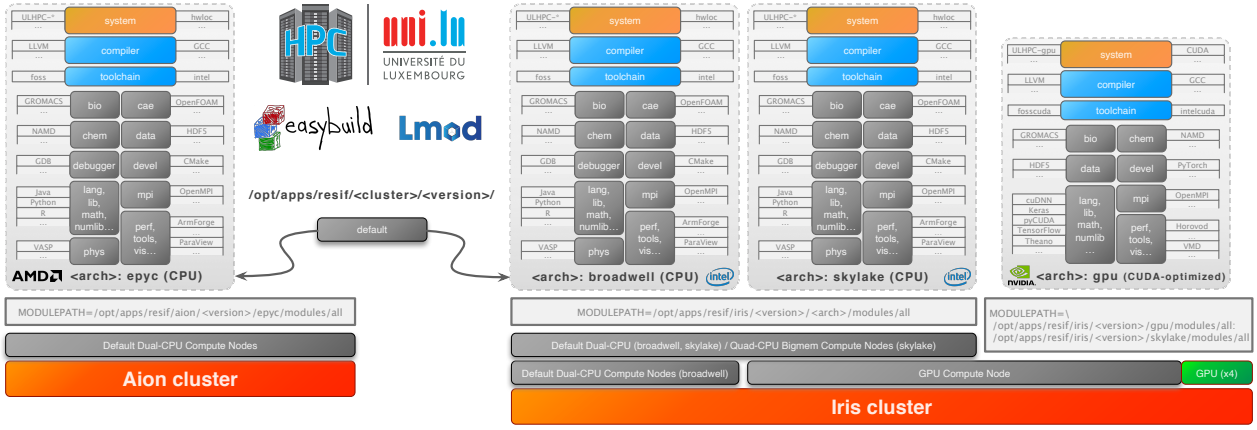

Figure 2: ULHPC software modules organization

From these settings, ULHPC modules and their associated software are kept organised through the categorized naming scheme. They are thus hosted within dedicated directories reflecting the cluster and the architecture the applications were compiled against for optimised builds, *i.e.*, `$RESIF_ROOT_DIR/<cluster>/<version>/<arch>/`. In particular, the production-ready modules and software are organized as depicted in *Figure 2* where the `MODULEPATH` environment variable is set to hold only one of these directories, and a `default` environment is defined as a symbolic link targeting the commonly supported CPU architecture for a given cluster (Ex: `broadwell` for our Intel-based Iris cluster). This ensures the most compliant default environment. A notable exception concerns the GPU computing nodes for which the accelerated CUDA-optimised scientific applications are kept as first searched modules, but completed by CPU-optimised builds (`skylake` in our case) to offer the most flexible environment. Then software sets are no longer defined as YAML files. Instead we now rely on native EasyBuild Bundles associated with a given toolchain version and organized by software category as reported in *Table 2*. This permits to fix for a given release the major software dependency components abstracted as `local_<name>ver` variables in the bundles which facilitates the porting of a given bundle definition from one version to another. *Table 3* summarizes the components version set for the current software set releases.

Building from scratch a complete software set takes quite a long time: around 184h for the 2019b release on the `skylake` CPU architecture. Easybuild supports the Slurm Scheduler to submit separate jobs and set dependencies between them to ensure they are run in the order dictated by the software dependency graph(s). Our experience with this natively supported feature was not demonstrating an expected reduction of the build time as it seems that the job dependencies are resolved in a linear fashion bringing little benefit to the parallel submission. For this reason, we are currently experimenting the reshaping of the bundles software dependency graph under the form of a *dominance tree* [3] structure where the nodes hold the software within the dependency graph of the bundle, and edges reflect a build dependency. This permits to quickly identify intermediate dominating software which can be built within concurrent jobs. Experience from the past software set builds can be used to assign weights to each node reflecting the expected built time. A static scheduling allows to set the job dependencies in Slurm to minimize the completion time of the full building process. Performance evaluations are in progress and demonstrate the interest of the approach – they will be reported in an expanded version of this article.

Finally, one of the most annoying limitations in the previous workflow was the explosion to custom easyconfigs, initially motivated by the necessity to tailor the default build process proposed by the community to the ULHPC computing node characteristics and specific Slurm and MPI bindings. Yet after only 3 software set generations (2017 to 2019), 565 "custom" easyconfigs were managed, and very quickly the divergence with the streamline EasyBuild developments led to a dead-end. This error-prone workflow was not repeated in RESIF 3.0. Taking advantage of the Github integration where the different EasyBuild repositories are located, specialized scripts were defined allowing to: (1) *create new Pull-Requests (PR)* for custom easyconfigs after checking the code style, and storing in the control repository the information related to the pending pull-request in `easyconfigs/pull-requests/<ID>`. Additional scripts permits to quickly complement a given pull-request with test reports executed on the HPC facility; (2) *update local easyconfigs from PR commits* that might be proposed by reviewers to correct the submitted easyconfigs. The full process is automated from authorized queries to the GitHub REST API; (3) *closing a merged Pull Request*

Table 2: ULHPC Bundles Overview

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...

which permits to cleanup the control repository from the custom easyconfigs no longer needed as they are now part of the streamline developments. Again, the same API is used to automate this operation. In addition, to prevent any future divergence from streamline developments, most of the customization is outsourced through special *hooks* which are in fact python callback functions that can be called during the different execution steps of Easybuilds. We selected the `parse_hook` function to inject on the fly extras parameters in the loaded easyconfig *before* their processing to simulate changes embedded in the original easyconfigs files. It permits to handle transparently installation keys and license definitions within the generated modules, when those settings are inherently specific to our site and are not meant to be exposed. These combined approaches (Github integration and generic hooks) allowed for more active contributions to the worldwide EasyBuild community while confining the custom configurations to the strict minimum *i.e.,* mainly optimised MPI suits or some commercial/internal software recipes. In practice, we have reduced by around 90% the number of custom easyconfigs in the RESIF control repository.

3 CONCLUSION

The increased complexity of HPC software coupled with an enhanced diversity in the dimension of computing environments featuring different processor architectures and generations, or the eventual presence of GPU accelerators remains a challenge. In this paper, we report the latest developments around RESIF 3.0, our internal framework aiming at the flexible, automated and consistent management of the scientific software sets deployed at our HPC facility. Designed as a wrapper around EasyBuild and LMod [5], the complete code refactoring effort mitigated previously identified pitfalls and led to a more versatile tool facilitating *reproducible* and *self-contained* deployment of the complete user software environment, coupled with a strong versioning policy favouring yearly release cycles in line with streamline developments. The proposed concepts are already demonstrating promising results and RESIF 3.0 is already increasing operational efficiency in production at the University of Luxembourg. A regression testing framework based on ReFrame [6], not reported in this article, is also rendering our framework more robust. In all cases, the software management techniques implemented in RESIF are applicable to a broad range of HPC facilities and thus may help other HPC centres to consolidate their own software management stack.

Table 3: ULHPC software set releases characteristics, mostly aligned with EasyBuild toolchains release. (*: *projections*)

	Software set release <version>			
Toolchain Component	**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

REFERENCES

[1] O. Ben-Kiki, C. Evans, and B. Ingerson. 2009. YAML Ain't Markup Language.
[2] R. H. Castain, J. Hursey, A. Bouteiller, and D. Solt. 2018. PMIx: Process management for exascale environments. *Parallel Comput.* 79 (2018), 9–29.
[3] R. Falke, R. Klein, R. Koschke, and J. Quante. 2005. The Dominance Tree in Visualizing Software Dependencies. In *3rd IEEE Intl. W. on Visualizing Software for Understanding and Analysis*. IEEE, Budapest, Hungary, 1–6.
[4] T. Gamblin, M. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, and S. Futral. 2015. The Spack package manager: bringing order to HPC software chaos. In *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Austin, TX, USA, 1–12.
[5] M. Geimer, K. Hoste, and R. McLay. 2014. Modern Scientific Software Management Using EasyBuild and Lmod. In *2014 First International Workshop on HPC User Support Tools*. IEEE, New Orleans, LA, USA, 41–51. https://doi.org/10.1109/HUST.2014.8
[6] S. Khuvis, Z-Q. You, H. Na, S. Brozell, E. Franz, T. Dockendorf, J. Gardiner, and K. Tomko. 2019. A Continuous Integration-Based Framework for Software Management. In *Proc. of the Practice and Experience in Advanced Research Computing (PEARC'19)*. ACM, New York, NY, USA, 1–7.
[7] D. Matthews and W. Limberg. 2018. MkDocs: documentation with Markdown. mkdocs.org.
[8] R. Mc.Lay. 2013. LMod: A New Environment Module System. https://lmod.rtfd.io.
[9] PuppetLabs. 2015. Puppet Hiera. https://puppet.com/docs/hiera/.
[10] S. Varrette, P. Bouvry, H. Cartiaux, and F. Georgatos. 2014. Management of an Academic HPC Cluster: The UL Experience. In *Proc. of the 2014 Intl. Conf. on High Performance Computing & Simulation (HPCS 2014)*. IEEE, Bologna, Italy, 959–967.