EOSC NORDIC

# D5.3 Virtual Computing Laboratories - SIGMA2 Report Public (M24 = September 2021)

| | |
|---|---|
| Author(s) | Andrey Kutuzov (AK), Sabry Razick (SR), Stephan Oepen (SO), Abdulrahman Azab (AA) |
| Status | Final |
| Version | 1.0 |
| Date | 3 September 2021 |

| Document identifier: | |
|---|---|
| Deliverable lead | AK (till the end of April 2021) |
| Related work package | |
| Author(s) | AK,SR,SO,AA |
| Contributor(s) | AK,SR,SO,AA |
| Due date | 3 September 2021 |
| Actual submission date | |
| Reviewed by | Ilja Livenson (WP3), Adil Hasan (WP5) |
| Approved by | |
| Dissemination level | Public |
| Website | https://source.coderefinery.org/nlpl/easybuild |
| Call | |
| Project Number | |
| Start date of Project | |
| Duration | |
| License | CC-BY-4.0 |
| Keywords | |



EOSC-Nordic project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857652

EOSC NORDIC

## Abstract:

This deliverable presents a method to provide the researchers with a consistent software environment across multiple computing facilities. The consistency is achieved by constructing a set of build scripts that will install the software tools with the same configurations and versions with the same set of dependencies. EasyBuild was selected as the framework to install and configure the software from source code, optimised for the underlying system and expose them as software-modules with uniform naming across systems. This way the researchers can bring their analysis pipelines and run on any of the systems just by loading the same set of modules.



EOSC-Nordic project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857652

www.eosc-nordic.eu

EOSC-Nordic project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857652

# Introduction

The Nordic Language Processing Laboratory (NLPL) is a collaboration of university research groups in Natural Language Processing (NLP) in Northern Europe with a vision to implement a virtual laboratory for large-scale NLP research. In this project a number of software packages and software pipelines are used. As the participants come from different institutes and use different computational services, the possibility to install these pieces of software in a uniform way has been a requirement. The aim of this sub-task as part of the NLPL use case in EOSC-Nordic is to investigate the suitability of the EasyBuild[1] build system.

We would like to organize provisioning of software for NLP research in a manner that makes it possible and cost-efficient to maintain the exact same software stack on multiple systems. Here, systems initially mean different High Performance Computer (HPC) systems, e.g. Puhti in Finland and Saga in Norway; in 2021, we anticipate to additionally support the LUMI[2] environment.

# Description of the platform components

## EasyBuild

EasyBuild is a software build and installation framework that allows you to manage (scientific) software on HPC systems in an efficient way. Our experience shows its suitability for the deployment of reproducible software environments for complex NLP tasks across different HPC systems, including multi-GPU and multi-node setups. It has also been successfully used in teaching deep learning for NLP in 2021 by the Language Technology Group at the University of Oslo[3]

## Toolchains (compiler toolchains)

A typical toolchain consists of one or more compilers, usually put together with some libraries for specific functionality, e.g., for using an MPI stack for distributed computing, or which provide optimized routines for commonly used math operations, e.g., the well-known BLAS/LAPACK/MKL APIs for linear algebra routines. We have selected the following tool chains:

[1] "EasyBuild." https://easybuild.io/. Accessed 3 Jun. 2021.
[2] "LUMI consortium - LUMI - LUMI supercomputer." https://www.lumi-supercomputer.eu/lumi-consortium/. Accessed 3 Jun. 2021.
[3] "IN5550 – Neural Methods in Natural Language Processing ... - UiO." https://www.uio.no/studier/emner/matnat/ifi/IN5550/. Accessed 3 Jun. 2021.



EOSC-Nordic project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857652

| Tool chain | Components |
|---|---|
| gomkl | GCC, OpenMPI, Intel Math Kernel Library (IMKL) 2019.1.144 and Intel FFTW wrappers |
| foss | GCC, OpenMPI, OpenBLAS 0.3.7 |

# Deliverables and Procedures

## Overall description

The NLPL virtual lab is technically a set of so called easyconfigs: description files (with the *.eb extension) which EasyBuild uses to actually build and deploy the corresponding software pieces as loadable modules. Modules are dependent on each other and accompanied by a set of convenience scripts and instructions. All the files related to this project are available as a git repository at:

https://source.coderefinery.org/nlpl/easybuild

Modules which are directly NLP-related are named with the “nlpl-” prefix, for example:

*nlpl-nvidia-bert-tf-20.06.08-gomkl-2019b-Python3.7.4.*

All the modules are provided in two mutually exclusive versions: for the *foss* toolchain and the *gomkl* toolchain (these strings are always in the name of the module). If a system is equipped with AMD CPUs, *foss* is the only option; if a system is equipped with Intel CPUs, one can try both toolchains. As a rule, *gomkl* is somewhat faster than *foss* for typical NLP loads.

## Rationale

In the approach described here, software pieces with their dependencies are compiled from source code on the system it will be running. This is in contrast to using pre-compiled binary installations (including standard Python wheels and conda packages). This allows fully enabling architecture-specific optimizations and a broad choice of optimized libraries (e.g. Intel MKL). As a result, one observes better performance in comparison. However, this process is cumbersome due to:

1. The need to locate the source code of the software components;
2. The need to discover all the dependencies so the software can be compiled and run on any system, regardless of its current configuration.
3. Steps 1 and 2 need to be performed recursively for all the dependencies;
4. The need to figure out the configuration options and compilation flags;
5. The need to test the installation;
6. The need to deploy the software in a consistent manner and need to construct a method for different versions of the software to live side-by-side;
7. The need for a way to present the available software components and versions to users so that the correct software package could be selected;
8. Once selected set the users runtime environment in an automatic and consistent way so that all



EOSC-Nordic project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857652

expected components are at users disposal without additional steps; The need for a method to avoid conflicting components being loaded at the same time.

The EasyBuild system addresses all the above issues by providing a method to create an automated build process with configuration files with block scripts that handle them. Once the configuration files and easyblocks are created, the end user needs only to invoke install command, EasyBuild will take care of the rest.

The following benefit can be seen in this approach to other solutions, like providing a prebuilt container image or a precompiled binary distribution.

1. This allows to fully enable architecture-specific optimizations;
2. Possibility for the users to make changes or try with different versions of dependencies without rebuilding everything as an image.
3. . Install dependencies in userspace instead of systemwide, preventing the need for root access and changes that might affect other users.
4. If distributing a container image or a container build file, someone needs to install the software at least once at some point. Thus EasyBuild can be a part of this containerization process, while the opposite is not true. Which means that when started by creating a method to properly install the software, it makes it possible to move to a distribution method like Docker or Conda later on. .It is difficult to build the process flow from a patched up container image, but if the process flow is created first then you can build a container using it.

5. The analysis environment, more often than not, involves multiple software packages. EasyBuild allows users to use multiple software packages at the same time and make sure that only compatible versions are loaded.
6. Solutions like Docker or Singularity depend heavily on those software being provided by the hostsystem. Even when they are provided, modifying a container image may not be possible on the system due to permission policies. Which means the user needs to do any modifications on a personal laptop and bring in the images only for runtime.
7. Container only solutions would require some knowledge on for example, how to access data from within a container and mount the home directory etc.

At the same time EasyBuild has the following disadvantages compared to Container based solutions.

1. When compiling on some systems there might be unexpected issues which never appear in the system where the process was designed. When using a container there is no need to compile it on the host system so this issue never arises.
2. Container images are easily disseminated. If there is an update to EasyBuild installed software, this needs to be compiled on all the systems it is installed on.
3. EasyBuild build system assumes that the host system has some module system installed and configured (e.g. Lmod)
4. Long term reproducibility on all systems not guaranteed when using the EasyBuild approach. For example many easybuild procedures do not work on ARM processor,



EOSC-Nordic project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857652

NLPL Virtual Laboratory is packaged to meet the needs of a typical NLP user as much as possible. We also separate different packages from each other: for example, TensorFlow 1.15.2 is not dependent on any particular SciPy version. This means that one can seamlessly upgrade SciPy without having to rebuild TensorFlow. In most cases, the installation of a particular piece of software is completely straightforward and automated. For example, the following command will install TensorFlow 2.3.2 for the gomkl toolchain and all its required dependencies:

*eb --robot easyconfigs/nlpl-tensorflow-2.3.2-gomkl-2019b-cuda-10.1.243-Python-3.7.4.eb*

All the necessary source archives will be downloaded automatically. Note that some software packages require manually downloading binary blobs from their websites after registration. Virtual Laboratory cannot do this, so the user is responsible for putting the required archives in the *blobs* subdirectory.

### Containers

So far, NLPL has shied away from using containers as the sole solution, in part simply because of lacking support on some of the target systems (notably Puhti and Mahti[4]), in part because of a concern for reduced transparency from the user point of view. Also, containerizing individual software modules severely challenges modularization: there is no straightforward way to 'mix and match' multiple containers into a uniform process environment, due to the fact that each container has its own file system. NLPL has dependency trees between modules

However, provisioning the *full NLPL* software (and possibly data) environment inside a container may offer some benefits, for example compatibility with cloud environments, increased uniformity across different systems, and potentially longer-term reproducibility. In this view, modularization would be obtained within the container, just as it does in the current environments on, for example, Puhti[5], Saga[6]. At the same time, the current solution where the software environment is setup using the EasyBuild framework can be used as part of the container image generation process.

## Storage requirements

NLPL Virtual Laboratory is essentially a set of lightweight easyconfigs, a set of modules created from them, and a set of software packages built according to these easyconfigs. Fully compiled laboratory (with versions of software for both *foss* and *gomkl* toolchains) takes up approximately 20 Gigabytes of disk space.

### Testing recipe

We have prepared a step-by-step guide to deploy the NLPL Virtual Laboratory on any HPC cluster. After all the steps are done, the user has a system which is able to train a toy BERT[7] model on a small Norwegian dataset using GPUs at hand.

The full recipe is available at http://wiki.nlpl.eu/index.php/Eosc/pretraining/nvidia.

## Author contributions

The following table summarises the specific contributions by each author.

[4] https://docs.csc.fi/computing/overview/
[5] https://research.csc.fi/-/puhti
[6] https://documentation.sigma2.no/hpc_machines/saga.html
[7] https://github.com/google-research/bert



EOSC-Nordic project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857652

| Participants | Activity | Description |
|---|---|---|
| AA | Strategic planning | Long term plan and integration with other work packages |
| SO | Create Git repository | For collaborative development and issue tracking. https://source.coderefinery.org/nlpl/easybuild |
| SR | Investigate a tool chain to be used used | Selecting an appropriate tool chain upfront is important for the following reasons.<br>1. Make sure all dependencies explicitly use the same compiler tool chain, so the tool chain can be replaced without ambiguity.<br>2. The compilers used should be available to all participants without any restriction like license and vendor locking.<br>3. Should include optimizations routines for math operations.<br>4. Should support parallel executions in shared and distributed memory setups. |
| SR, AK, SO | Create EasyBuild procedure for selected software | EasyBuild procedure was developed for Tensorflow 2.3.2 and Tensorflow 1.15.2 with all dependencies, using foss and gomkl tool chains |
| AK | Create test cases and performance comparison | Make sure the intended performance enhancements are active and performance can be compared across systems. The results of our benchmark tests are available at http://wiki.nlpl.eu/index.php/Eosc/easybuild/benchmark |
| AK | A common stack of NLP modules in a fully automated EasyBuild configuration | http://wiki.nlpl.eu/index.php/Eosc/pretraining/nvidia |
| AK | Installing on Saga and Puhti | http://wiki.nlpl.eu/Infrastructure/software/easybuild |



EOSC-Nordic project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857652

# NLPL Virtual Laboratory modules

Following table lists the modules built during this deliverable. dules (except toolchains) exist in two versions: for the *foss* and *gomkl* toolchains. The module name is in accordance with the EasyBuild naming convention and shows the name of the software, version and the tool chain used. All modules would use the CPUs present in the system in an optimal way, where the module is installed. Some modules as indicated would use the GPUs if the GPUs are present on the system and would fall back to using CPUs if absent. Modules compiled with OPenMPI would use the message passing interface to use distributed memory, which means it can utilize multiple computer nodes during runtime

| Module name | Uses CPU | Uses GPU | Uses OpenMPI | Role |
|---|---|---|---|---|
| imkl-2019.1.144-gompi-2019b | v | | v | Intel Math Kernel Library required for the *gomkl* toolchain |
| gomkl-2019b | v | | v | *gomkl* toolchain itself |
| cuDNN-7.6.4.38-CUDA-10.1.243 | v | v | | GPU library for deep learning |
| NCCL-2.6.4-CUDA-10.1 | v | v | v | Multi-device GPU training drivers |
| nlpl-cython-0.29.21 | v | | v | Required for Numpy |
| nlpl-numpy-1.18.1 | v | | v | Central array-processing library for Python |
| nlpl-scipy-ecosystem-2021.01 | v | | v | A collection of mathematical software[8] typically used in NLP: *scipy*, *matplotlib*, etc |
| nlpl-python-candy-2021.01 | v | | | Various small Python packages required for other modules |
| nlpl-nlptools-2021.01 | v | | | Growing collection of small NLP tools without many dependencies. Currently includes *conllu* and *seqeval*. |
| nlpl-gensim-3.8.3 | v | | v | Popular Python library for operations with word embeddings and topic |

[8] https://scipy.org/



EOSC-Nordic project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857652

| | | | | |
|---|---|---|---|---|
| | | | | modeling |
| nlpl-nltk-3.5 | v | | | Popular Python library for NLP operations. Includes data repository with corpora, datasets, treebanks and lexical databases. |
| nlpl-bazel-0.26.1 | v | | | Building system, required for TensorFlow |
| nlpl-bazel-3.4.1 | v | | | Building system, required for TensorFlow |
| nlpl-h5py-2.10.0 | v | | v | Python interface for HDF5 |
| nlpl-h5py-3.1.0 | v | | v | Python interface for HDF5 |
| nlpl-horovod-0.20.3 | v | v | v | Distributed training framework for TensorFlow and PyTorch |
| nlpl-tensorflow-1.15.2 | v | v | v | TensorFlow 1 |
| nlpl-dllogger-0.1.0 | v | | | Logging tool, required for NVIDIA BERT implementation |
| nlpl-nvidia-bert-tf-20.06.08 | v | v | v | NVIDIA BERT implementation |
| nlpl-tensorflow-2.3.2 | v | v | v | TensorFlow 2 |
| nlpl-pytorch-1.6.0 | v | v | v | PyTorch |
| nlpl-tokenizers-0.10.2 | v | | | Fast tokenization library, required for Transformers |
| nlpl-transformers-4.5.1 | v | v | v | HuggingFace Transformers, current tool of choice for NLP practitioners. |

## Summary

In the research software world, it is natural that prototype software and software pipelines developed and used are sometimes tightly coupled to local systems, without much planning on distributing them. This was the case with The Nordic Language Processing Laboratory (NLPL) group as well. For example, they had pipelines setup on the Abel compute cluster at University of Oslo,



EOSC-Nordic project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857652

which they needed to migrate away after it was decommissioned. During this project we have investigated methods to make these pipelines portable. Initial suggestion was to focus on container technology like Docker. The three main issues with that approach were identified. First, when creating a container, the software still needs to be installed and configured, this involves locating, downloading the software, preprocessing, locating and acquiring dependencies, configuring the correct compilation environment and compilation. Secondt, to avoid the first caveat, if a static container image is composed from a combination of precompiled binaries, system packages and manual tweakings that can not be reproduced, we will endup with a static image that would take a tremendous effort, even to make a very slight change. Third, we do not have a consistent container policy across HPC systems, which makes it difficult to create one recipe for all.

Therefore, the strategy described here to create a set of EasyBuild recipes to commision the pipelines was followed. . This involved creating or adaptingEasyBuild recipes that describe the requirements mentioned above as the first reason for not using container strategy as the only solution. The recipes are distributed as a deliverable hosted on CodeRefinery GitLab service[9]. If EasyBuild is configured on the target system, these recipes will automatically perform all steps from locating the software to creating a module file, when loaded sets the users path correctly to point to the installation. In addition as compatible dependency trees created for different pipelines users can mix them as it suits. At the same time, different versions of the same software can be installed alongside and the users can select which one is active at a given instance. The installed software would be optimized for the underlying system as well, which makes the shared resource usage optimal, especially on HPC systems.

If the EasyBuild is not configured on the target system, it is possible to fabricate a container recipe with EasyBuild as a component and then use the EasyBuild recipes to take care of the rest of the process. Another option is to maintain a conda repository with precompiled platform specific binaries, that are created using the EasyBuild recipes developed during this deliverable.

At the end we have demonstrated a portable pipeline distribution strategy, without restricting ourselves to one technology. This deliverable thus solves the portability issue of the pipelines used by The Nordic Language Processing Laboratory (NLPL) and facilitates realizing their vision to implement a virtual laboratory for large-scale NLP research.

We would like to organize provisioning of software for NLP research in a manner that makes it possible and cost-efficient to maintain the exact same software stack on multiple systems. Here, systems initially mean different High Performance Computer (HPC) systems, e.g. Puhti in Finland and Saga in Norway; in 2021, we anticipate to additionally support the LUMI[10] environment.

[9] https://source.coderefinery.org/

[10] "LUMI consortium - LUMI - LUMI supercomputer." https://www.lumi-supercomputer.eu/lumi-consortium/. Accessed 3 Jun. 2021.



EOSC-Nordic project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857652

EOSC NORDIC

www.eosc-nordic.eu

EOSC-Nordic project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857652