Working with EasyBuild

EasyBuild is a build and installation framework that facilitates management of (scientific) software. All GNU/Linux workstations and servers have a variety of softwares available through our EasyBuild environment. Because EasyBuild is a powerful framework that enables everybody to build and install complex softwares with ease, we encourage users to use it in their development workflows. Here we give a few tips should you decide to employ EasyBuild in your private software stack. In no circumstances the information that follows should replace the official EasyBuild manual.

Common EasyBuild Environments

Our EasyBuild environments and softwares are accessible to any GNU/Linux server and workstation users according to the following schemas

Where	OS	Mount Point	Remote Location	Protocol
Workstations & Servers	Fedora 33	/easybuild/easybuild/fc31	Software Server	NFS
	RHEL7	/easybuild/easybuild/el7	Software Server	NFS
	RHEL8	/easybuild/easybuild/el8	Software Server	NFS
Xmaris	CentOS 7	/marisdata/easybuild	Marisdata	NFS

These provide a variety of softwares including all EasyBuild command-line tools such as `eb` via software modules. A list of available software can be displayed via `module spider`.

To make EasyBuild's tools available under your environment use

```
module load EasyBuild
```

Sometimes, it could be better to have EasyBuild available on a workstation locally and not remotely to avoid network-related bottlenecks. In this case, a fresh installation of EasyBuild to a local disk can be performed following these instructions. Pay, however, particular attention to perform the installation in a disk where there is enough space available and for which you have writing access.

Build and Install Software via EasyBuild

Preliminary Setup and Considerations

First of all you need to setup your EasyBuild development stack. This will be hosted in a location on your server/workstation for which you have writing access. We first make EasyBuild available under our environment, then we define the location of our EasyBuild software stack `/path/to/your/easybuild/stack` and ultimately we prepend to the `MODULEPATH` the path in which our private-software stack modules will be installed. Using the bash syntax

```
module load EasyBuild
```

```
export EASYBUILD_PREFIX=/path/to/your/easybuild/stack

module use $EASYBUILD_PREFIX/modules/all
```

If you want to use the softwares installed in your private stack on a variety of hardwares (workstations and servers) you must also instruct EasyBuild to build hardware-independent executables

```
export EASYBUILD_OPTARCH=GENERIC
```

Failing to do so, can result in the production of non-portable softwares. On the other hand, we advise you build hardware-bound softwares in all cases in which execution performance is paramount.

Put particular attention if you are planning to build *OpenBLAS* via EasyBuild. In this case defining EASYBUILD_OPTARCH=GENERIC is not sufficient to produce portable software (CPU independent). Use both `export EASYBUILD_OPTARCH=GENERIC` in your setup and `–try-amend=buildopts='TARGET=CORE2 DYNAMIC_ARCH=1 DYNAMIC_OLDER=1 BINARY=64 USE_THREAD=1 USE_OPENMP=1 CC=“$CC” FC=“$F77”'` as EasyBuild (`eb`) runtime option. If your compilation fails you can try `–try-amend=buildopts='TARGET=CORE2 BINARY=64 USE_THREAD=1 USE_OPENMP=1 CC=“$CC” FC=“$F77”'` instead.

If you are building OpenMPI on a cluster whose resources are managed by Slurm and you would like to use slurm's `srun` (instead of `mpirun` or `mpiexec`) to run parallel applications, then you must configure OpenMPI to do so via the `eb` runtime option `–try-amend=configopts=“–enable-mpi1-compatibility –with-slurm –with-pmi=/usr –with-pmi-libdir=/usr/lib64 CPPFLAGS=-I/usr/include/slurm LDFLAGS=-L/usr/lib64 ”`. Clearly, adapt this line to the exact location of the slurm libraries and headers on your cluster.

Build your first software via EasyBuild

Now that you have setup your EasyBuild development environment, you can search the EasyBuild software repository for softwares you would like to install. Here we search for EasyBuild software configurations (or easyconfigs) whose name starts with `Miniconda`

```
eb -S ^Miniconda
CFGS1=/easybuild/easybuild/fc31/software/EasyBuild/4.1.2/easybuild/easyconfigs/m
 * $CFGS1/Miniconda2/Miniconda2-4.3.21.eb
 * $CFGS1/Miniconda2/Miniconda2-4.6.14.eb
 * $CFGS1/Miniconda2/Miniconda2-4.7.10.eb
 * $CFGS1/Miniconda3/Miniconda3-4.4.10.eb
 * $CFGS1/Miniconda3/Miniconda3-4.5.12.eb
 * $CFGS1/Miniconda3/Miniconda3-4.6.14.eb
 * $CFGS1/Miniconda3/Miniconda3-4.7.10.eb
```

To install `Miniconda3-4.7.10` and any needed dependencies (`-r` option) type

```
eb -r Miniconda3-4.7.10.eb
```

The software will be installed in `/path/to/your/easybuild/stack/software` and its corresponding module needed to make it available in your environment in `/path/to/your/easybuild/stack/modules/all`.

At this point you can use your newly installed `Miniconda3` software by sourcing its module via

```
module load Miniconda3
which conda
```

Please note that EasyBuild gives its modules names that follow a particular scheme based on the easyconfigs that generated them. If you are not sure of the module name, you can always consult the output of `module avail miniconda`.

Of particular importance are the following EasyBuild `eb` runtime options, but you are encouraged to consult `eb –help`

Option	Explanation
–dry-run	Print build overview incl. dependencies (full paths) (default: False)
–dry-run-short	Print build overview incl. dependencies (short paths) (default: False)
–extended-dry-run	Print build environment and (expected) build procedure that will be performed (default: False)
–rebuild	Rebuild software, even if module already exists (don't skip OS dependencies checks) (default: False)
–robot=PATH[:PATH]	Enable dependency resolution, using easyconfigs in specified paths (type pathsep-separated list; default: EasyBuild installation dir)
–skip	Skip existing software (useful for installing additional packages) (default: False)

Build a toolchain via EasyBuild

An EasyBuild toolchain is a set of softwares that consists of one or more compilers and some libraries that have a specific aim, e.g., for performing parallel computations on an HPC cluster or for using Graphical Processing Units (GPUs). In oder words, you will be able to install a set of softwares for a specific functionality with just one command.

List wich toolchains are available via

```
eb --list-toolchains
List of known toolchains (toolchainname: module[,module...]):
    ClangGCC: Clang, GCC
    CrayCCE: PrgEnv-cray
    CrayGNU: PrgEnv-gnu
    CrayIntel: PrgEnv-intel
    CrayPGI: PrgEnv-pgi
    GCC: GCC
    GCCcore: GCCcore
    GNU: GCC
    PGI: PGI
    cgmpich: Clang, GCC, MPICH
    cgmpolf: BLACS, Clang, FFTW, GCC, MPICH, OpenBLAS, ScaLAPACK
    cgmvapich2: Clang, GCC, MVAPICH2
```

```
    cgmvolf: BLACS, Clang, FFTW, GCC, MVAPICH2, OpenBLAS, ScaLAPACK
    cgompi: Clang, GCC, OpenMPI
    cgoolf: BLACS, Clang, FFTW, GCC, OpenBLAS, OpenMPI, ScaLAPACK
    foss: BLACS, FFTW, GCC, OpenBLAS, OpenMPI, ScaLAPACK
    fosscuda: BLACS, CUDA, FFTW, GCC, OpenBLAS, OpenMPI, ScaLAPACK
    gcccuda: CUDA, GCC
    gimkl: GCC, imkl, impi
        ...
```

If you wanted to install the `foss` (Free and Open Source Software) toolchain first analyse the output of `eb -S ^foss` to see which easyconfigs provide you which *foss* version and then execute for instance

```
eb -r foss-2019b.eb
```

Once the installation process is terminated, you will have `BLACS, FFTW, GCC, OpenBLAS, OpenMPI, ScaLAPACK` installed in your software stack.

Build a software for which no easyconfig is available

This is an advanced topic and requires some extra information on how EasyBuild builds and installs a given software. So far we have seen that it is straightforward to install a software from a given easyconfig file. But what to do if EasyBuild does not provide in its repos an easyconfig for the software you would like to install? Read on.

Easyblocks

EasyBuild installations hinge on the concept of easyblocks. An easyblock is a basic unit of installation. There are easyblocks that performs *configure/make/make install* or just *pip install* to build and install softwares. A complete list of available easyblocks is given by the output of `eb –list-easyblocks`. Easyblocks are written in python. For example if you wanted to install a custom software via the common workflow *configure/make/make install* you would use the `ConfigureMake` easyblock.

Easyconfigs

Because easyblocks only offer the basic build and install functionality for a specific software, it is often needed to customise them according to the installation task in progress. This is done via easyconfig files. These are python files which inherit the behavior of a specific easyblock and customise its behaviour via the modification of specific parameters. There are common parameters to all easyblocks and parameteres that are specific to a particular easyblocks. See here.

Build and install from custom easyconfig

As you might have inferred, in all cases in which EasyBuild does not provide in its repos an easyconfig for the software you would like to install, you will have to pick up the *right* easyblock and write an *ad-*

hoc easyconfig file which uses the chosen easyblock with appropriate parameters. This task is not simple. To make things more difficult, there could be cases in which you will have to write your own easyblock from scratch! Here follows an example easyconfig that will install a combo (bundle) of python packages all available in a single module.

```
cat Quantum-TensorFlow-2.1.0-foss-2019b-Python-3.7.4.eb
#
easyblock = 'PythonBundle'

name = 'Quantum-TensorFlow'
version = '2.1.0'
versionsuffix = '-Python-%(pyver)s'

homepage = 'https://www.tensorflow.org/'
description = "An open-source software library for Machine Intelligence with
some quantum software"

toolchain = {'name': 'fosscuda', 'version': '2019b'}
toolchainopts = {'usempi': True, 'pic': True}

dependencies = [
    ('Python', '3.7.4'),
    ('TensorFlow', '2.1.0',versionsuffix,('fosscuda','2019b')),
]
exts_default_options = {
    'source_urls': [PYPI_SOURCE],
    'sanity_pip_check': True,
}
use_pip = True

exts_list = [
    ('PubChemPy', '1.0.4', {
        'checksums':
['24e9dc2fc90ab153b2764bf805e510b1410700884faf0510a9e7cf0d61d8ed0e'],
    }),

    ('openfermion', '0.11.0', {
        'checksums':
['2aede7cf2e5f7be4c0016c9b542c27505644f8ecb9411c653dc89a5cd746f84c'],
    }),

    ('cirq', '0.8.0', {
       'source_tmpl': 'cirq-0.8.0-py3-none-any.whl',
       'unpack_sources': False,
       'checksums':
['f424f042ec058cf9e5dd993050bd22b850470019dca57e337a2e3d0a2e416265'],
   }),
]

sanity_check_commands = [
    'python -c "import tensorflow as tf; from openfermion.ops import
```

```
FermionOperator, QubitOperator"'
]




moduleclass = 'lib'
```

Apart from the self-explicative instructions given in the file above, note the following

- We define a list of build and runtime dependencies via the list `dependencies`
- All python softwares are installed as *extensions* via `pip` by means of `use_pip`
- All extensions (python packages) are sourced from PyPi and their details is given in the list `exts_list`
- The build/install process will succeed only if `sanity_check_commands` exit without errors

Now install it via

```
ls my_easyconfigs
Quantum-TensorFlow-2.1.0-foss-2019b-Python-3.7.4.eb
eb -r Quantum-TensorFlow-2.1.0-foss-2019b-Python-3.7.4.eb
```

Write a custom easyblock

In the unlikely event that no suitable easyblocks fit your software installation procedure, you will have to implement your own easyblock.

Here follows a trivial – perhaps not very useful – example in which we create an easyblock that implements the following function: it prints a screen message when its corresponding module is loaded. This example should get you started and give you an idea of how easyblocks work.

```
# cat anacondaleonardo.py
from easybuild.easyblocks.a.anaconda import EB_Anaconda

class AnacondaLeonardo(EB_Anaconda):
    """Support for building/installing Anaconda and Miniconda."""
    def make_module_extra(self):
        txt = super(AnacondaLeonardo, self).make_module_extra()
        txt += self.module_generator.msg_on_load("Use at your own risk, I
shall assume no responsabilities.")
        return txt
```

Notice the following

- easyblocks are written in python
- we are modifying the behaviour of the EasyBuild-provided `EB_Anaconda` easyblock by subclassing it
- this easyblock can therefore be used to install the anaconda software
- we override the function `make_module_extra` to adapt it to our needs

Now create en easyconfig file that uses the newly created easyblock

```
#cat Miniconda2-4.3.21_mod.eb
easyblock = 'AnacondaLeonardo'

name = 'Miniconda2'
version = '4.3.21'

homepage = 'https://docs.conda.io/en/latest/miniconda.html'

description = """Miniconda is a free minimal installer for conda. It is a
small,
bootstrap version of Anaconda that includes only conda, Python, the packages
they
depend on, and a small number of other useful packages.

A warning message will be printed on the screen upon module loading.

Author: leonardo

"""

toolchain = SYSTEM

source_urls = ['https://repo.anaconda.com/miniconda/']
sources = ['%(name)s-%(version)s-Linux-x86_64.sh']
checksums =
['5097d5ec484a345c8785655113b19b88bfcd69af5f25a36c832ce498f02ea052']

moduleclass = 'lang'
```

And install it via `eb -r /path/to/easyconfig/Miniconda2-4.3.21_mod.eb –include-easyblocks=/path/to/my/easyblocks/*py`.

Finally, do not forget to read the official easyblocks documentation.

EasyBuild Tips

Read the docs

Always read the official documentation relative to the version you are using. These pages are not meant to substitute it.

Heterogeneous environments

If you are planning to use your EasyBuild-built software on a variety of CPUs, do not forget to instruct EasyBuild to do so via `export EASYBUILD_OPTARCH=GENERIC` and `eb … –try-`

`amend=buildopts='TARGET=CORE2 DYNAMIC_ARCH=1 DYNAMIC_OLDER=1 BINARY=64 USE_THREAD=1 USE_OPENMP=1 CC="$CC" FC="$F77"'` as EasyBuild (eb) runtime option if your are building OpenBLAS [1].

Do you want to know on what hardware you are? `gcc -march=native -Q –help=target | awk '/march/{print $2}'`

Learn from examples

Always consult existing EasyBuild recipes and learn from them. `grep -ri pythonbundle /easybuild/easybuild/fc31/software/EasyBuild/*/easybuild/easyconfigs` on a workstation will return a list of easyconfigs from which you can learn all sorts of tricks that concern the `pythonbundle` easyblock.

Environment conflicts

Should any preset environment variables conflict with the correct execution of a program or should you want to modify the environment at all, you can do this directly in your easyconfig file. In the example below, we show how to *unset* the environment variable LD_LIBRARY_PATH upon module loading

```
#...
modtclfooter = "unsetenv LD_LIBRARY_PATH"
modluafooter = 'unsetenv("LD_LIBRARY_PATH")'
#...
```

Python extensions

When you install a python package as an extension, EasyBuild checks if the extension is working properly by *python-importing* the extension name. This means that for extensions such as PyYAML, the building process will fail because no module exists named PyYAML. You can overcome the default behaviour by either giving the extension a custom `modulename`

```
 ('PyYAML', '5.3.1', {
        'checksums':
['b8eac752c5e14d3eca0e6dd9199cd627518cb5ec06add0de9d32baeee6fe645d'],
        'modulename': 'yaml',
    }),
```

or by skipping it altogether (dangerous)

```
 ('PyYAML', '5.3.1', {
        'checksums':
['b8eac752c5e14d3eca0e6dd9199cd627518cb5ec06add0de9d32baeee6fe645d'],
        'modulename': False,
    }),
```

If the module you are installing contains a lot of extensions, its rebuild process could last a long time if we were to re-install the software package and all its extensions from scratch. EasyBuild fortunately has an option that let us install one or more additional extensions without having to reinstall the software package and all extensions from scratch. This saves a considerable amount of time

```
eb my_easyconfig.eb --skip --rebuild
```

Read about partial installations.

GPUs

If you are building software with GPU support, do so on a workstation/server with GPUs and specify the CUDA compute capability of the attached GPU(s), for instance `eb … –cuda-compute-capabilities=6.0`.

TensorFlow

If you want to customise the https://www.tensorflow.org/TensorFlow building process you must know that TensorFlow installations occur via Bazel. This means that a whole lot of customisations can take place at the Bazel level. At building time, Bazel will source `$HOME/.bazelrc` which you could use to manipulate the installation at your convenience, for instance

```
# cat ~/.bazelrc
build -c opt
build --cxxopt="-O3"
build --cxxopt="-march=native"
build –cxxopt="D_GLIBCXX_USE_CXX11_ABI=0"
# and so on
```

A common case in which such manipulations are needed is for the installation of TensorFlow Ops. An Op will work only if it was built in the same way as TensorFlow itself. So sometimes it is necessary to rebuild TensorFlow or the Op to have a matching building process.

1)
https://easybuild.readthedocs.io/en/latest/Controlling_compiler_optimization_flags.html#build-environment-vs-hardcoding-in-build-scripts