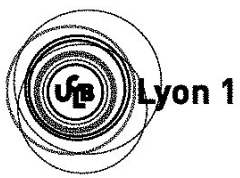
EasyBuild : introduction

Anne Cadiou

Laboratoire de Mécanique des Fluides et d'Acoustique

Ateliers et Séminaires Pour l'Informatique et le Calcul Scientifique
PMCS2I - LMFA
Jeudi 27 novembre 2025



Introduction

EasyBuild est un outil de construction et d'installation de logiciels scientifique dans le contexte HPC

Caractéristiques

- installe les logiciels, bibliothèques et leurs dépendances
- compile depuis les sources
- optimise pour l'architecture cible
- produit des modules
- fournit des recettes de construction
- open source

Principale alternative
`https://spack.io/`

Prescriptif

principe de chaîne d'outils : la version du compilateur détermine la version des dépendances

Un peu d'histoire

Conçu en 2008 par l'équipe HPC de l'université de Gand

- Première version annoncée à Supercomputing 2012
- Framework communautaire, utilisé par différents centres HPC (LUMI, Jülich, CSCS, ...)

Suite ?

Projet EESSI, européen, mêmes développeurs qu'EasyBuild
`https://www.eessi.io/`

Installation

Pré-requis

- une distribution Linux
- des outils du shell (tar, bzip, rpm, dpkg, ulimit, etc.)
- Python (>= 3.6 est requis pour EasyBuild >=5.0)
- un outil de modules : modules d'environnement Tcl(/C) ou Lmod
- un compilateur C et C++ (par exemple, gcc et g++)

cf. `https://docs.easybuild.io/installation`

Exemple

Laptop sous Ubuntu 22.04

Charger environnement module

```
sudo apt-get install environment-modules
```

Variables d'environnement associées

```
  . /usr/share/modules/init/bash
```

ou ajouter dans le `~/.bashrc`

```
if ! shopt -q login_shell; then
    . /usr/share/modules/init/bash
fi
```

Vérifier la version de Python utilisée

```
python3 --version

  python 3.10.12
```

Définir la version pour EasyBuild

```
export EB_PYTHON=python3.10
```

Par défaut, les sources sont téléchargées dans `~/.local/`

Installer EasyBuild avec EasyBuild

Création de l'environnement Python

```
python3 -m venv venv-eb
source venv-eb/bin/activate
pip install easybuild
```

Dans l'environnement virtuel se trouve

```
easybuild==5.1.2
easybuild-easyblocks==5.1.2
easybuild-easyconfigs==5.1.2
easybuild-framework==5.1.2
```

Le déploiement d'EB se fait par une commande du type

```
eb --install-latest-eb-release \
  --modules-tool=EnvironmentModules --module-syntax=Tcl \
  --prefix=/home/acadiou/demo-eb/tools \
  --buildpath=/dev/shm/softs/eb/build \
  --repositorypath=/home/acadiou/easybuild/eb/ebfiles_repo \
  --sourcepath=/home/acadiou/easybuild/eb/sources
```

où la première ligne est l'installation, la deuxième ligne spécifie l'environnement module et les suivantes donnent éventuellement des variables d'environnement.

Configuration

Par défaut, l'ensemble des fichiers s'installe dans `~/.local/`
Pour définit d'autres répertoire, il est pratique d'utiliser un fichier de configuration
`~/.config/easybuild/config.cfg`
Obtention de la syntaxe

```
mkdir -p $HOME/.config/easybuild
eb --confighelp > $HOME/.config/easybuild/config.cfg
```

Variables d'environnement à modifier

```
[config]
prefix=/home/acadiou/demo-eb/tools
buildpath=/dev/shm/softs/eb/build
repositorypath=/home/acadiou/easybuild/eb/ebfiles_repo
sourcepath=/home/acadiou/easybuild/eb/sources
parallel=8
modules-tool=EnvironmentModules
module-syntax=Tcl
```

Positionner la variable `EASYBUILD_CONFIGFILES` pour lire la configuration

```
export EASYBUILD_CONFIGFILES=$HOME/.config/easybuild/config.cfg
```

Déploiement

Par défaut

```
ls $prefix/
  build ebfiles_repo modules software sources
```

et ici

```
ls $prefix
  modules software

ls /dev/shm/softs/eb
  build

ls /home/acadiou/easybuild/eb
  ebfiles_repo sources
```

Noter que

```
ls $prefix/modules/all
  EasyBuild
```

Module EasyBuild

Chargement d'EasyBuild

```
module use ${prefix}/modules/all
```

Vérification de la configuration

```
which eb
  demo-eb/venv-eb/bin/eb

module load EasyBuild
module list

  Currently Loaded Modulefiles:
    1) EasyBuild/5.1.2

which eb
  tools/software/EasyBuild/5.1.2/bin/eb
```

```
eb --version
```

```
eb --show-config
```

Utilisation

Lister les chaînes de version

```
eb --list-toolchains
```

Le déploiement de modules s'effectue à partir de fichiers `easyconfigs` qui spécifient avec les options de recherche, l'ensemble de la procédure

- le block d'easybuild auquel il corresponds
- la toolchain
- l'origine des sources à télécharger et compiler
- les options de compîlation
- les dépendances
- les tests
- les logs

Les fichiers préconfigurés ont pour suffixe `.eb` et se trouvent dans

- en local dans `EasyBuild/5.1.2/easybuild/easyconfigs`

Options utiles

Recherche de logiciel

```
eb -S ^FFT
```

renvoie la liste des fichiers `easyconfigs` existants

```
eb FFTW.MPI-3.3.10-gompi-2025b.eb -Dr
```

`-r, --robot` recherche les dépendances

`-x, --extended-dry-run` permet d'afficher toutes les étapes, combiné à `-r`

`-D, --dry-run` sans exécuter

`-M, --missing` liste les dépendances manquantes

Installe de (très) nombreux modules, qui peuvent être exposés ou masqués (déployés en modules cachés, ce qui nécessite de faire attention à ne pas générer de doublons, si on veut optimiser l'espace disque).

Fichier d'installation

```
name = 'FFTW.MPI'
version = '3.3.10'

homepage = 'https://www.fftw.org'
description = """FFTW is a C subroutine library for computing the discrete Fourier
      transform (DFT)
in one or more dimensions, of arbitrary input size, and of both real and complex
     data."""

toolchain = {'name': 'gompi', 'version': '2025b'}
toolchainopts = {'pic': True}

source_urls = [homepage]
sources = ['fftw-%(version)s.tar.gz']
checksums = ['56c932549852cddcfafdab3820b0200c7742675be92179e59e6215b340e26467']

dependencies = [('FFTW', '3.3.10')]

runtest = 'check'

moduleclass = 'numlib'
```

S'obtient avec

```
eb --show-ec FFTW.MPI-3.3.10-gompi-2025b.eb
```

Description du fichier EasyBuild

Nom et version du logiciel

- `name = 'FFTW.MPI'`
  Le nom du logiciel à installer est FFTW.MPI. FFTW (Fastest Fourier Transform in the West) est une bibliothèque bien connue pour le calcul de transformées de Fourier discrètes (DFT). La version MPI indique qu'il s'agit d'une version parallèle utilisant l'interface MPI (Message Passing Interface) pour le calcul distribué.
- `version = '3.3.10'`
  La version spécifique de FFTW à installer est 3.3.10.

Informations Générales

- homepage = 'https ://www.fftw.org'
  L'URL du site officiel de FFTW, où l'on peut trouver plus d'informations et la documentation.
- description = """..."""
  Une description détaillée de FFTW : il s'agit d'une bibliothèque en C pour calculer des transformées de Fourier discrètes (DFT) en une ou plusieurs dimensions, pour des tailles d'entrée arbitraires, et pour des données réelles ou complexes.

Toolchain et options

- `toolchain = 'name': 'gompi', 'version': '2025b'`
  La toolchain est un ensemble d'outils de compilation et de bibliothèques utilisés pour construire le logiciel. Ici, la toolchain gompi (GCC + OpenMPI) version 2025b est spécifiée. Cela signifie que le logiciel sera compilé avec le compilateur GCC et la bibliothèque MPI OpenMPI.
- `toolchainopts = 'pic': True`
  L'option PIC (Position Independent Code) est activée. Cela permet de générer du code indépendant de la position en mémoire, ce qui est utile pour créer des bibliothèques partagées.

Sources et vérification

- `source_urls = [homepage]`
  L'URL de téléchargement des sources est la même que la page d'accueil.
- `sources = ['fftw-%(version)s.tar.gz']`
  Le nom du fichier source à télécharger, ici, `fftw-3.3.10.tar.gz`
- `checksums = ['56c932549852(...)b340e26467']`
  La somme de contrôle (checksum) du fichier source, utilisée pour vérifier l'intégrité du fichier téléchargé.

Dépendances

- `dependencies = [('FFTW', '3.3.10')]`

  FFTW.MPI dépend de la version non-MPI de FFTW (version 3.3.10). Cela signifie que la bibliothèque FFTW classique doit être installée avant FFTW.MPI.

Tests et classification

- `runtest = 'check'`

  Après l'installation, EasyBuild exécutera la commande make check pour vérifier que le logiciel fonctionne correctement.

- `moduleclass = 'numlib'`

  Le module EasyBuild sera classé dans la catégorie numlib (bibliothèques numériques), ce qui facilite son organisation et sa recherche dans l'environnement des modules.

Options

Il est possible d'utiliser l'option `--try-toolchain-version` pour tester une autre chaîne que celle proposée.

```
eb -S ^BWA |grep GCCcore
```

```
 * $CFGS1/BWA/BWA-0.7.17-20220923-GCCcore-12.3.0.eb
 * $CFGS1/BWA/BWA-0.7.17-GCCcore-11.2.0.eb
 * $CFGS1/BWA/BWA-0.7.17-GCCcore-11.3.0.eb
 * $CFGS1/BWA/BWA-0.7.17-GCCcore-12.2.0.eb
 * $CFGS1/BWA/BWA-0.7.17-GCCcore-12.3.0.eb
 * $CFGS1/BWA/BWA-0.7.18-GCCcore-12.3.0.eb
 * $CFGS1/BWA/BWA-0.7.18-GCCcore-13.2.0.eb
 * $CFGS1/BWA/BWA-0.7.18-GCCcore-13.3.0.eb
 * $CFGS1/BWA/BWA-0.7.19-GCCcore-14.2.0.eb
 * $CFGS1/bwa-mem2/bwa-mem2-2.2.1-GCCcore-12.3.0.eb
```

et avec `GCCcore-15.2.0` cela donnerait

```
eb BWA-0.7.19-GCCcore-14.2.0.eb --try-toolchain-version 15.2.0 -Dr
```

Solution proposée dans

```
/tmp/eb-29g8d8st/tweaked_easyconfigs/BWA-0.7.19-GCCcore-15.2.0.eb
```

Exercice

- Trouver les versions de `cmake` disponibles
- Choisir une version et afficher les dépendances sans exécuter l'installation
- Trouver avec quelle version de `flex` le module est construit
- Trouver avec quelles options de compilation le `make` est construit

Corrigé : trouver les versions de cmake disponibles

```
eb -S ^cmake
```

```
== found valid index for /home/acadiou/ASPICS/Cours_EasyBuild/demo-eb/tools/
     software/EasyBuild/5.1.2/easybuild/easyconfigs, so using it...
CFGS1=/home/acadiou/ASPICS/Cours_EasyBuild/demo-eb/tools/software/EasyBuild/5.1.2/
     easybuild/easyconfigs/c/CMake
 * $CFGS1/CMake-3.18.4.eb
 * $CFGS1/CMake-3.20.1-GCCcore-10.3.0.eb
 * $CFGS1/CMake-3.21.1-GCCcore-11.2.0.eb
 * $CFGS1/CMake-3.22.1-GCCcore-11.2.0.eb
 * $CFGS1/CMake-3.23.1-GCCcore-11.3.0.eb
 * $CFGS1/CMake-3.24.3-GCCcore-11.3.0.eb
 * $CFGS1/CMake-3.24.3-GCCcore-12.2.0.eb
 * $CFGS1/CMake-3.26.3-GCCcore-12.3.0.eb
 * $CFGS1/CMake-3.26.3-GCCcore-13.1.0.eb
 * $CFGS1/CMake-3.27.6-GCCcore-13.2.0.eb
 * $CFGS1/CMake-3.29.3-GCCcore-13.3.0.eb
 * $CFGS1/CMake-3.31.3-GCCcore-14.2.0.eb
 * $CFGS1/CMake-3.31.8-GCCcore-12.3.0.eb
 * $CFGS1/CMake-3.31.8-GCCcore-13.3.0.eb
 * $CFGS1/CMake-3.31.8-GCCcore-14.3.0.eb
 * $CFGS1/CMake-3.31.8.eb
 * $CFGS1/CMake-4.0.3-GCCcore-14.3.0.eb
```

Corrigé : afficher les dépendances sans exécuter l'installation

```
eb CMake-4.0.3-GCCcore-14.3.0.eb -Dr
```

```
Dry run: printing build status of easyconfigs and dependencies
CFGS=/home/acadiou/ASPICS/Cours_EasyBuild/demo-eb/tools/software/EasyBuild/5.1.2/
     easybuild/easyconfigs
 * [x] $CFGS/m/M4/M4-1.4.20.eb (module: M4/1.4.20)
 * [x] $CFGS/z/zlib/zlib-1.3.1.eb (module: zlib/1.3.1)
 * [x] $CFGS/m/M4/M4-1.4.19.eb (module: M4/1.4.19)
 * [x] $CFGS/b/Bison/Bison-3.8.2.eb (module: Bison/3.8.2)
 * [x] $CFGS/f/flex/flex-2.6.4.eb (module: flex/2.6.4)
 * [ ] $CFGS/b/binutils/binutils-2.44.eb (module: binutils/2.44)
 * [ ] $CFGS/g/GCCcore/GCCcore-14.3.0.eb (module: GCCcore/14.3.0)
 * [ ] $CFGS/h/help2man/help2man-1.49.3-GCCcore-14.3.0.eb (module: help2man
      /1.49.3-GCCcore-14.3.0)
```

(...)

```
 * [ ] $CFGS/p/Python/Python-3.13.5-GCCcore-14.3.0.eb (module: Python/3.13.5-
      GCCcore-14.3.0)
 * [ ] $CFGS/l/libpsl/libpsl-0.21.5-GCCcore-14.3.0.eb (module: libpsl/0.21.5-
      GCCcore-14.3.0)
 * [ ] $CFGS/c/cURL/cURL-8.14.1-GCCcore-14.3.0.eb (module: cURL/8.14.1-GCCcore
      -14.3.0)
 * [ ] $CFGS/c/CMake/CMake-4.0.3-GCCcore-14.3.0.eb (module: CMake/4.0.3-GCCcore
      -14.3.0)
== Temporary log file(s) /tmp/eb-4p6icrlo/easybuild-d5_7x2d3.log* have been
     removed.
== Temporary directory /tmp/eb-4p6icrlo has been removed.
```

Corrigé : trouver avec quelle version de `flex` le module est construit

```
eb --show-ec CMake-4.0.3-GCCcore-14.3.0.eb
```

```
toolchain = {'name': 'GCCcore', 'version': '14.3.0'}
```

```
builddependencies = [
    ('binutils', '2.44'),
]

dependencies = [
    ('ncurses', '6.5'),
    ('zlib', '1.3.1'),
    ('bzip2', '1.0.8'),
    ('cURL', '8.14.1'),
    ('libarchive', '3.8.1'),
    ('OpenSSL', '3', '', SYSTEM),
]
```

```
eb CMake-4.0.3-GCCcore-14.3.0.eb -Dr |grep flex
```

```
 * [x] $CFGS/f/flex/flex-2.6.4.eb (module: flex/2.6.4)
 * [ ] $CFGS/f/flex/flex-2.6.4-GCCcore-14.3.0.eb (module: flex/2.6.4-GCCcore
      -14.3.0)
```

`flex` n'est pas une dépendance directe de `CMake` mais de `GCCcore`.

Corrigé : trouver avec quelles options de compilation le `make` est construit

```
eb CMake-4.0.3-GCCcore-14.3.0.eb -x > log.log
```

```
export CC='gcc'
export CFLAGS='-O2 -ftree-vectorize -march=native -fno-math-errno'
export CPPFLAGS=''
export CXX='g++'
export CXXFLAGS='-O2 -ftree-vectorize -march=native -fno-math-errno'
export F77='gfortran'
export F90='gfortran'
export F90FLAGS='-O2 -ftree-vectorize -march=native -fno-math-errno'
export FC='gfortran'
export FCFLAGS='-O2 -ftree-vectorize -march=native -fno-math-errno'
export FFLAGS='-O2 -ftree-vectorize -march=native -fno-math-errno'
export FLIBS='-lgfortran'
export LDFLAGS=''
export LIBS='-lm -lpthread'
export OPTFLAGS='-O2 -ftree-vectorize -march=native'
export PRECFLAGS='-fno-math-errno'
```

Utilisation des modules installés par EasyBuild

Une fois les modules installés, seules les commandes `module` comptent.

```
. /usr/share/modules/init/bash
module use tools/modules/all
```

Commandes standard

```
module purge
module avail (-a)
module list
module load bzip2
```

Génération de fichier d'installation

Il est possible de modifier les fichiers, pour les adapter à des nouvelles versions de compilateur, corriger des erreurs.
Des erreurs peuvent se produire, liées

- aux dépendances (modules manquants, incompatibles, etc.)
- au fichier de recette (syntaxe incorrecte, variables non définies, chemin mal spécifié, etc.)
- aux droits d'écriture sur des répertoires cibles (`/opt`, `/usr/local`, etc.)
- variables d'environnement manquantes (`EASYBUILD_PREFIX`, etc.)
- à des conflits avec des logiciels déjà installés
- aux sources (introuvables, corrompues, téléchargement interrompu, etc.)
- patch manquants
- à l'échec de la compilation (incompatibilités matétrielles, logicielles, etc.)
- à la version obsolète d'EasyBuild
- à des interruptions réseau, etc.

Utilisation de patch

Les patchs sont souvent utilisés pour corriger des bugs, adapter le code source à un environnement spécifique, ou résoudre des problèmes de compatibilité.

Supposons qu'on veuille installer un logiciel nommé Foo-1.0, mais que son code source contient une erreur mineure dans un fichier Makefile : en effet, le fichier Makefile utilise un chemin absolu (`/usr/local/include`) pour inclure des en-têtes, mais il faut le rendre relatif. Solution : écrire le patch dans un fichier `Foo-1.0-Makefile.patch` qui contient

```
--- Foo-1.0/Makefile.orig 2025-11-15 10:00:00.000000000 +0100
+++ Foo-1.0/Makefile 2025-11-15 10:01:00.000000000 +0100
@@ -1,7 +1,7 @@
 # Makefile for Foo

-CFLAGS = -I/usr/local/include
+CFLAGS = -I$(FOO_INCLUDE_DIR)

 all: foo

 install: all
```

Ce patch remplace le chemin absolu `/usr/local/include` par une variable `$FOO_INCLUDE_DIR`, ce qui permet de configurer dynamiquement le chemin des en-têtes.

Intégration dans la recette

```
name = 'Foo'
version = '1.0'

homepage = 'https://example.com/foo'
description = """Foo is a sample software."""

toolchain = {'name': 'GCCcore', 'version': '11.3.0'}

source_urls = ['https://example.com/foo']
sources = ['Foo-1.0.tar.gz']
checksums = ['sha256': 'abc123...']

# Ajoute le patch
patches = ['Foo-1.0-Makefile.patch']

# Définis la variable d'environnement pour le patch
builddependencies = [
    ('FOO_INCLUDE_DIR', '/path/to/custom/include'),
]

builddir = 'Foo-%(version)s'

build_cmd = 'make'
install_cmd = 'make install'

moduleclass = 'tools'
```

Exercice sur les erreurs

Charger EasyBuild

```
. /usr/share/modules/init/bash
module use tools/modules/all
```

Essayer de faire une installation en utilisant ce fichier `zlib-trouble-1.2.13.eb`, qui contient deux erreurs.

```
easyblock = 'ConfigureMake'

name = 'zlib'
version = '1.2.13'

homepage = 'https://www.zlib.net/'

description = """
 zlib is designed to be a free, general-purpose, legally unencumbered -- that
 is, not covered by any patents -- lossless data-compression library for use
 on virtually any computer hardware and operating system.
"""

toolchain = {'name': 'PrgEnv-gnu', 'version': '24.02'}
```

(...)

```
checksums = ['b3a24de97a8fdbc835b9833169501030b8977031bcb54b3b3ac13740f846ab30']

# need to take care of $CFLAGS ourselves with SYSTEM toolchain
# we need to add -fPIC, but should also include -O* option to avoid
# compiling with -O0 (default for GCC)
buildopts = 'CFLAGS="-O2 -fPIC"'

sanity_check_paths = {
    'files': ['include/zconf.h', 'include/zlib.h', 'lib/libz.a',
              'lib/libz.%s' % SHLIB_EXT],
    'dirs': [],
}

moduleclass = 'lib'
```

Diagnostiquer les problèmes et les corriger.

Correction

```
eb ./zlib-trouble-1.2.13.eb
```

```
ERROR: Failed to process easyconfig /home/acadiou/ASPICS/Cours_EasyBuild/
     Presentation/scripts/zlib-trouble-1.2.13.eb: Toolchain PrgEnv-gnu not found,
     available toolchains: gcccuda,gompic,fosscuda,gomklc,goolfc,golfc,gimpic,
     giolfc,gmklc,iccifortcuda,iimklc,iimpic,intelcuda,iompic,iomklc,nvompic,
     nvpsmpic,system,GCCcore,GCC,gompi,foss,goalf,gobff,goblf,gofbf,gomkl,goolf,
     gfbf,golf,gimpi,gimkl,giolf,gmkl,gmvapich2,gmacml,gmvolf,gmpich,gmpflf,gmpolf
     ,gmpich2,gmpit,gpsmpi,gpsolf,gqacml,gsmpi,gsolf,ClangGCC,cgmpich,cgmpolf,
     cgmvapich2,cgmvolf,cgompi,cgoolf,GNU,iccifort,iimpi,ictce,iibff,intel,iimkl,
     iiqmpi,iqacml,impich,impmkl,ipsmpi,intel-para,iompi,iofbf,iomkl,ismkl,intel-
     compilers,ifbf,PGI,pmkl,pompi,pomkl,CrayGNU,CrayIntel,CrayPGI,CrayCCE,FCC,
     ffmpi,Fujitsu,NVHPC,nvompi,nvofbf,nvpsmpi,xlcxlf,xlmpich,xlmpich2,xlmvapich2,
     xlompi
```

Ce qui montre que le toolchain n'est pas présente.
Remplacer

```
toolchain = {'name': 'PrgEnv-gnu', 'version': '24.02'}
```

par

```
toolchain = SYSTEM
```

Relancer l'installation

```
eb ./zlib-trouble-1.2.13-correction01.eb
```

```
== configuring...
  >> running shell command:
  ./configure --prefix=/home/acadiou/ASPICS/Cours_EasyBuild/demo-eb/tools/software
       /zlib/1.2.13
  [started at: 2025-11-16 19:20:02]
  [working dir: /dev/shm/softs/eb/build/zlib/1.2.13/system-system]
  [output and state saved to /tmp/eb-op4o7shu/run-shell-cmd-output/configure-
       gbc6hg8x]

ERROR: Shell command failed!
```

Ce qui indique un problème avec les sources.
Leur emplacement est manquant. Ajouter

```
source_urls = ['https://zlib.net/fossils/']
sources = [SOURCELOWER_TAR_GZ]
```

et dans ce cas l'installation d'effectue jusqu'au bout.

Conclusion

- logiciel facile à prendre en main
- rapidité de mise à disposition de modules
- facilité de modification des recettes
- prescriptif en reposant sur des chaînes de compilation

Références

- https://easybuild.io/
- https://docs.easybuild.io/
- https://easybuild.io/tech-talk/
- https://tutorial.easybuild.io/
- https://lumi-supercomputer.github.io/easybuild-tutorial/
- https://www.eessi.io/