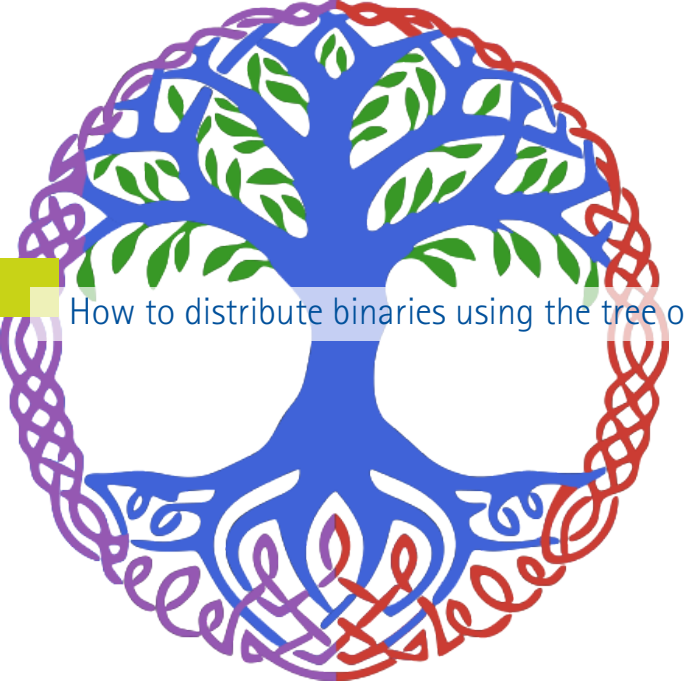


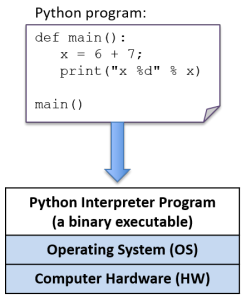
Leibniz Universität Hannover

How to distribute binaries using the tree of life

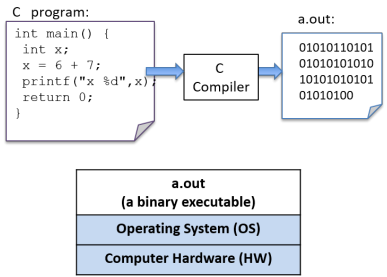
What is a binary (executable)?

A binary executable is...

- file with binary content (that is a sequence of 0s and 1s)



Python: Interpreted Execution



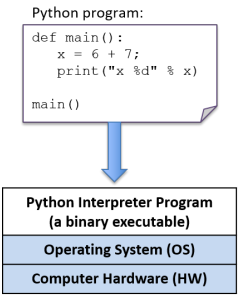
C: First compiled into a.out
Then direct execution of a.out

from https://diveintosystems.org/

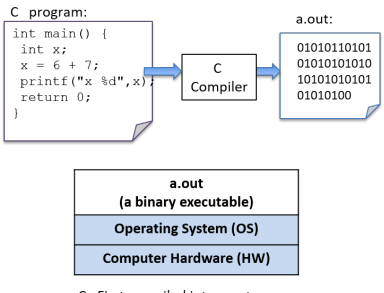
What is a binary (executable)?

A binary executable is...

- file with binary content (that is a sequence of 0s and 1s)
- directly executable by the target system, given it conforms to the systems ABI



Python: Interpreted Execution



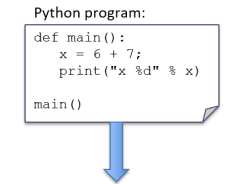
C: First compiled into a.out
Then direct execution of a.out

from https://diveintosystems.org/

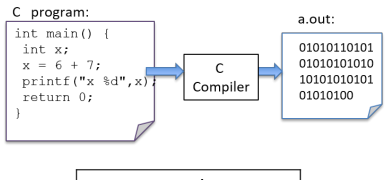
What is a binary (executable)?

A binary executable is...

- file with binary content (that is a sequence of 0s and 1s)
- directly executable by the target system, given it conforms to the systems ABI
- facilitating separation of product and source code



Python: Interpreted Execution



C: First compiled into a.out
Then direct execution of a.out

from https://diveintosystems.org/

Binary dependencies
Example

Leibniz Universität Hannover

Lets assume I write a julia program and want to reuse a program written in another language by calling its binary.
That is as easy as

```
julia> run(`fortune`)
Q:  Why did the tachyon cross the road?
A:  Because it was on the other side.
Process(`fortune`, ProcessExited(0))
```

Binary dependencies
Except...

- You don't know what version you will call
- Whether it's installed at all

```
julia> run(`fortune`)
ERROR: IOError: could not spawn `fortune`: no such file or directory (ENOENT)
```

Binary dependencies
Solutions of the past

2012 Julia pulicly announced

Binary dependencies
Solutions of the past

Leibniz Universität Hannover

2012 Julia pulicly announced

2013 `BinDeps.jl`: at `Pkg.build()` time binaries are downloaded/ build on-demand on the target machine

Binary dependencies
Solutions of the past

Leibniz Universität Hannover

2012 Julia pulicly announced

2013 `BinDeps.jl`: at `Pkg.build()` time binaries are downloaded/ build on-demand on the target machine

2017 `BinaryBuilder.jl` & `BinaryProvider.jl`: binaries get build in a sandbox environment and distributed decentrally by the creators

Binary dependencies
Solutions of the past

Leibniz Universität Hannover

2012 Julia pulicly announced

2013 `BinDeps.jl`: at `Pkg.build()` time binaries are downloaded/ build on-demand on the target machine

2017 `BinaryBuilder.jl` & `BinaryProvider.jl`: binaries get build in a sandbox environment and distributed decentrally by the creators

2019 `Artifacts.jl` & `Yggdrasil`: binaries get build in a sandbox environment on a dedicated build machine and will automatically provided by a julia wrapper package (JLL) centrally hosted under JuliaBinaryWrappers

Binary dependencies
End of all issues*

giordano on 2 Sep 2019 • edited

Contributor

This PR installs Cairo and Pango library using `BinaryProvider.jl`.

I think it's good to keep this open for some time to let users play with it. So, please ***do test this PR*** and report back any issue you face. To do this, just run

```
]add https://github.com/giordano/Cairo.jl.git#binary-builder
```

in Julia REPL. I recommend then to clean the directory `Cairo/deps/` from `build.log` and `deps.jl`, and then you can run `]build Cairo`.

Building these libraries has been a great stress-test for the new `BinaryBuilder.jl` framework that will land in Julia v1.3. A *huge* thanks to **@staticfloat**! I'll probably update the URLs of some `build.jl` files later this week (I'll use the "official" ones that will be in @JuliaBinaryWrappers), but the content of the downloaded tarballs should be the more or less the same.

When merged, this PR fixes #105, fixes #121, fixes #148, fixes #162, fixes #165, fixes #185, fixes #187, fixes #203, fixes #207, fixes #214, fixes #230, fixes #239, fixes #256, fixes #258, fixes #261, fixes #265, fixes #266, fixes #271, fixes #279, fixes #284, fixes #286, fixes #287. It supersedes #149, #196, #289.

1 20

Leibniz Universität Hannover

`BinaryBuilder.jl`

- provides an alpine linux based sandbox environment as `.squashfs`-image
- this environment has all tools bundled to cross-compile your program to the supported platforms
- it will tell all sorts of convenient lies to mimic the target platform (`uname`, `sysctl`, ...)
- sets appropiate environment variables and uses oldest libc possible for maximal compatibility
- will prevent assumptions (glidc vs musl, uses non-standard paths)
- currently supports C/C++, FORTRAN, Go, Rust

BinaryBuilder.jl
Compiler shards

Leibniz Universität Hannover

Specify the target platform in `architecture-OS-library` triplets.

```
julia> triplet.(supported_platforms())
16-element Vector{String}:
 "i686-linux-gnu"            "x86_64-apple-darwin"
 "x86_64-linux-gnu"          "aarch64-apple-darwin"
 "aarch64-linux-gnu"         "x86_64-unknown-freebsd"
 "armv6l-linux-gnueabihf"    "i686-w64-mingw32"
 "armv7l-linux-gnueabihf"    "x86_64-w64-mingw32"
 "powerpc64le-linux-gnu"
 "i686-linux-musl"
 "x86_64-linux-musl"
 "aarch64-linux-musl"
 "armv6l-linux-musleabihf"
 "armv7l-linux-musleabihf"
```

BinaryBuilder.jl
Accounting for incompabilities

Leibniz Universität Hannover

C++: `std::string` can have C++03 or C++11 string ABI

FORTRAN: `libgfortran` also has 3 different ABIs

Custom feratures can be used, e.g. to account for different microarchitectures (AVX2, AVX512, etc...)

- After building, there is an extra auditing step catching known portability issues

BinaryBuilder.jl
Building recipe I

Leibniz Universität Hannover

```
using BinaryBuilder

name = "libfoo"
version = v"1.0.1"
sources = [
    ArchiveSource("<url to source tarball>", "sha256 hash"),
]

script = raw"""
cd ${WORKSPACE}/srcdir/libfoo-*
make -j${nproc}
make install
"""

platforms = supported_platforms()
```

BinaryBuilder.jl
Building recipe II

Leibniz Universität Hannover

```
products = [
    LibraryProduct("libfoo", :libfoo),
    ExecutableProduct("fooifier", :fooifier),
]

dependencies = [
    Dependency("Zlib_jll"),
]

build_tarballs(ARGS, name, version, sources, script, platforms, products,
     dependencies)
```

This `build_tarballs.jl` can be built using the wizard via `BinaryBuilder.run_wizard()`.

BinaryBuilder.jl
fortune

Leibniz Universität Hannover

https://github.com/JuliaPackaging/Yggdrasil/pull/6625
https://github.com/JuliaRegistries/General/pull/82293
https://github.com/JuliaBinaryWrappers/fortune_jll.jl

```
pkg> add fortune_jll
julia> using fortune_jll
julia> run(`$(fortune())`)
Every cloud has a silver lining; you should have sold it, and bought titanium.
```

BinaryBuilder.jl
FAQ

Leibniz Universität Hannover

Hey, this is cool, can I use this for my non-Julia related project?

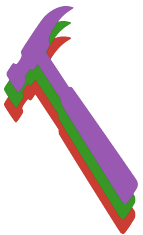
Absolutely! There's nothing Julia-specific about the binaries generated by the cross-compilers used by `BinaryBuilder.jl`.

BinaryBuilder.jl
FAQ

Leibniz Universität Hannover

Hey, this is cool, can I use this for my non-Julia related project?

Absolutely! There's nothing Julia-specific about the binaries generated by the cross-compilers used by `BinaryBuilder.jl`.



Happy building!