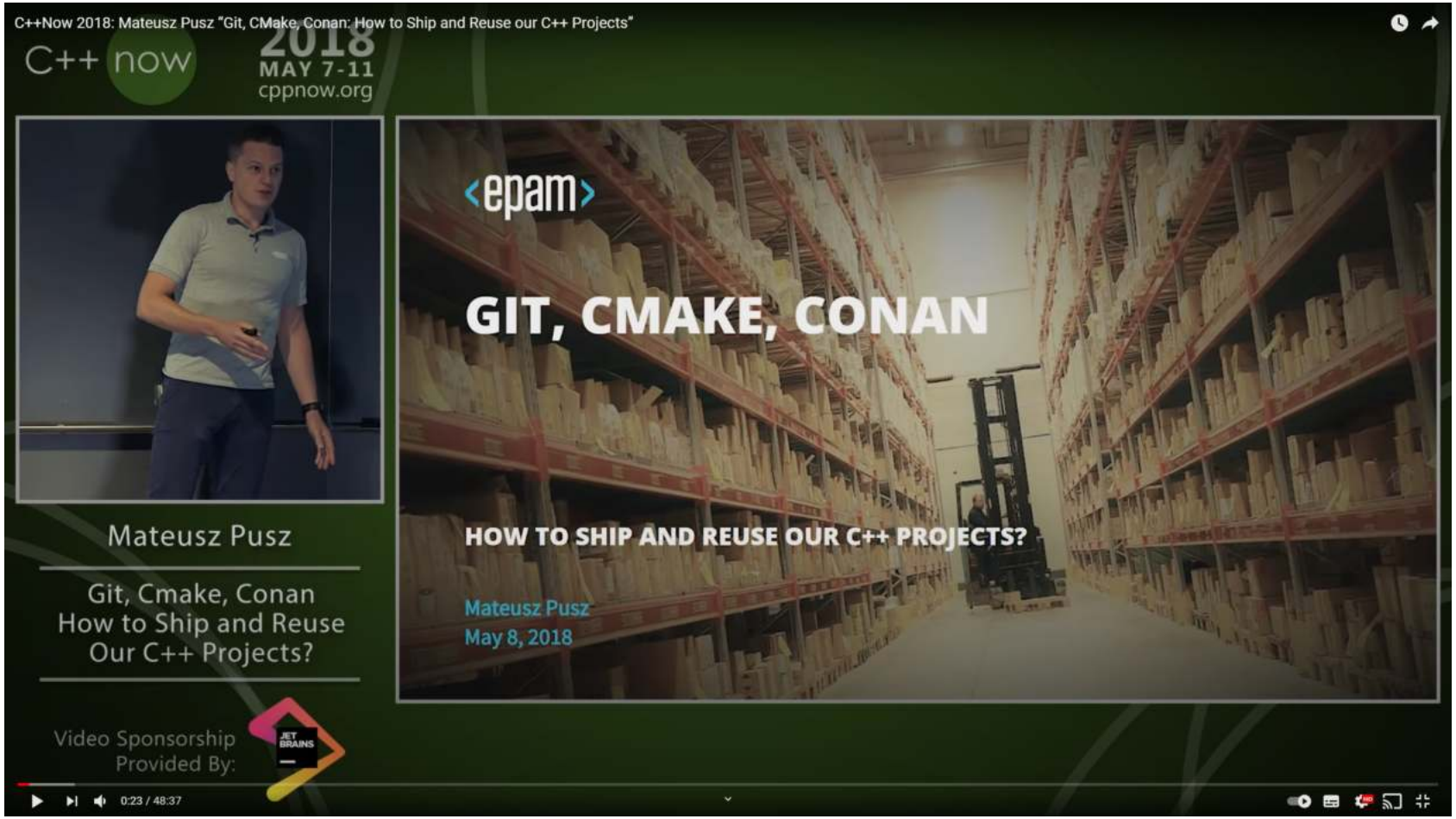
CMake + Conan: 3 Years Later

Mateusz Pusz

C++ now
2021
MAY 2-7
Aspen, Colorado, USA

C++Now 2018


C++Now 2018: Mateusz Pusz "Git, CMake, Conan: How to Ship and Reuse our C++ Projects"
C++ now
2018
MAY 7-11
cppnow.org
Mateusz Pusz
Git, Cmake, Conan
How to Ship and Reuse
Our C++ Projects?
Video Sponsorship
Provided By:
JET BRAINS
<epam>
GIT, CMAKE, CONAN
HOW TO SHIP AND REUSE OUR C++ PROJECTS?
Mateusz Pusz
May 8, 2018
0:23 / 48:37

What has changed?

PRODUCT	THEN	TODAY	RELEASES

What has changed?

PRODUCT	THEN	TODAY	RELEASES
CMake	3.11.1	3.20.1	64

What has changed?

PRODUCT	THEN	TODAY	RELEASES
CMake	3.11.1	3.20.1	64
Conan	1.3.2	1.36.0	123

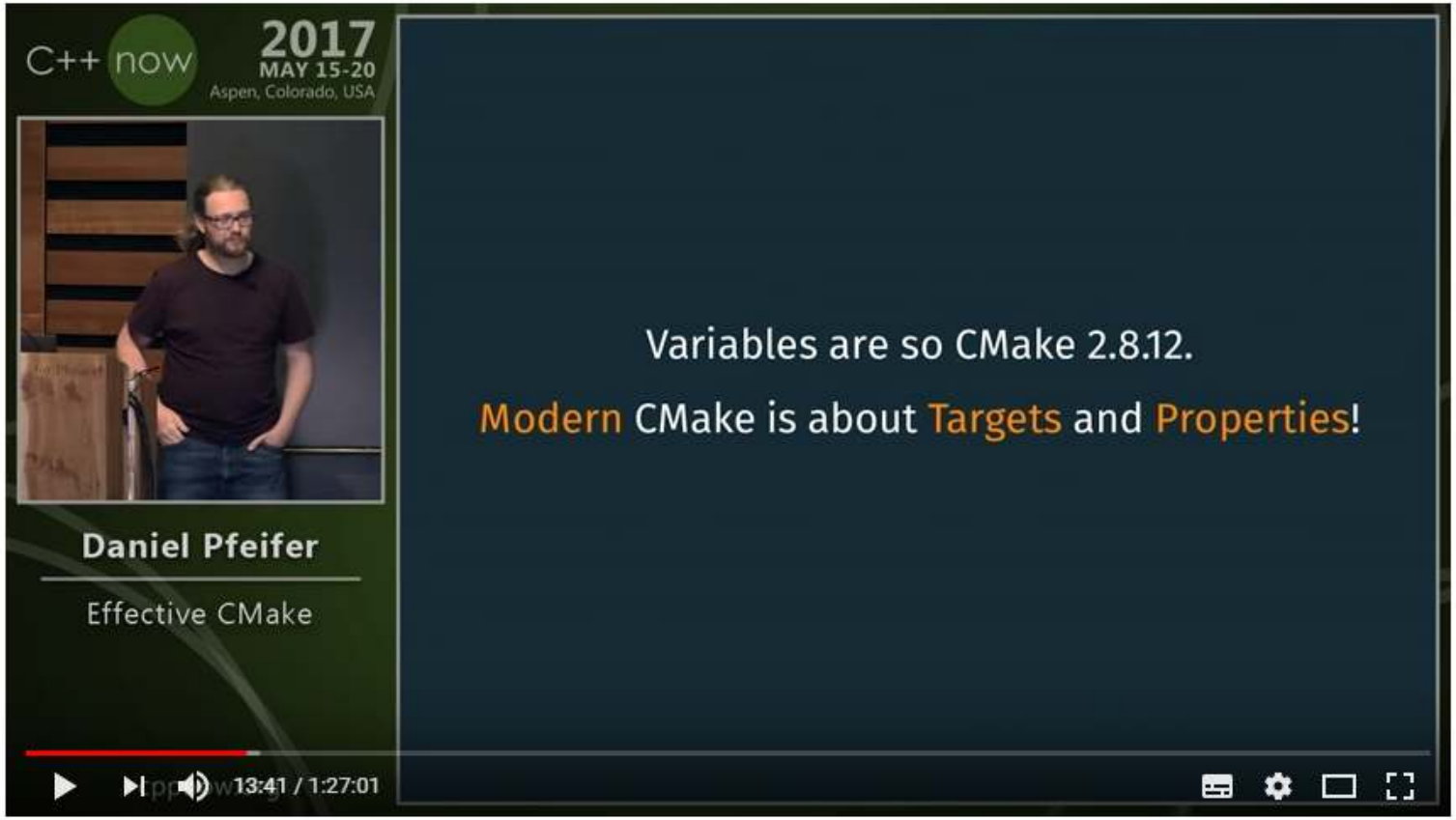
What has changed?

PRODUCT	THEN	TODAY	RELEASES
CMake	3.11.1	3.20.1	64
Conan	1.3.2	1.36.0	123
Me	39	42	???

CMAKE

CROSS-PLATFORM C++ BUILD GENERATOR
(NOT A GENERAL-PURPOSE SCRIPTING LANGUAGE)

Effective CMake



C++Now 2017: Daniel Pfeifer “Effective CMake"

Modern CMake

Don't introduce custom CMake variables unless you really need to.

- Custom variables
  - are **error prone**
  - often introduce a lot of noise to the code
  - **complicate maintenance**

Modern CMake

Don't introduce custom CMake variables unless you really need to.

- Custom variables
  - are **error prone**
  - often introduce a lot of noise to the code
  - **complicate maintenance**

CMake variables seem to be easy but **they are not**.
This is the biggest problem!

A Motivating Example (Poll #1)

What will be printed during the configuration phase?

```
set(foo 0)

message(foo)
message("foo")
message(${foo})

if(foo)
  message("#1")
endif()
if("foo")
  message("#2")
endif()
if(${foo})
  message("#3")
endif()
```

A Motivating Example (Poll #1)

What will be printed during the configuration phase?

```
set(foo 0)

message(foo)
message("foo")
message(${foo})

if(foo)
  message("#1")
endif()
if("foo")
  message("#2")
endif()
if(${foo})
  message("#3")
endif()
```

```
foo
foo
0
```

A Motivating Example (Poll #2)

What will be printed during the configuration phase?

```
set(foo ON)

message(foo)
message("foo")
message(${foo})

if(foo)
  message("#1")
endif()
if("foo")
  message("#2")
endif()
if(${foo})
  message("#3")
endif()
```

A Motivating Example (Poll #2)

What will be printed during the configuration phase?

```
set(foo ON)

message(foo)
message("foo")
message(${foo})

if(foo)
  message("#1")
endif()
if("foo")
  message("#2")
endif()
if(${foo})
  message("#3")
endif()
```

```
foo
foo
ON
#1
#3
```

A Motivating Example (Poll #3)

What will be printed during the configuration phase?

```
set(foo abc)

message(foo)
message("foo")
message(${foo})

if(foo)
  message("#1")
endif()
if("foo")
  message("#2")
endif()
if(${foo})
  message("#3")
endif()
```

A Motivating Example (Poll #3)

What will be printed during the configuration phase?

```
set(foo abc)

message(foo)
message("foo")
message(${foo})

if(foo)
  message("#1")
endif()
if("foo")
  message("#2")
endif()
if(${foo})
  message("#3")
endif()
```

```
foo
foo
abc
#1
```

A Motivating Example (Poll #4)

What will be printed during the configuration phase?

```
set(abc abc)
set(foo abc)

message(foo)
message("foo")
message(${foo})

if(foo)
  message("#1")
endif()
if("foo")
  message("#2")
endif()
if(${foo})
  message("#3")
endif()
```

A Motivating Example (Poll #4)

What will be printed during the configuration phase?

```
set(abc abc)
set(foo abc)

message(foo)
message("foo")
message(${foo})

if(foo)
  message("#1")
endif()
if("foo")
  message("#2")
endif()
if(${foo})
  message("#3")
endif()
```

```
foo
foo
abc
#1
#3
```

Basic Expressions

```
if(value)
```

- Quoted or unquoted constant with value **ON**, **YES**, **TRUE** or **Y** -> `true`
  - the test is case-insensitive
- Quoted or unquoted constant with value **OFF**, **NO**, **FALSE**, **N**, **IGNORE**, **NOTFOUND**, an empty string or a string that ends in **-NOTFOUND** -> `false`
  - the test is case-insensitive
- Number -> converted to a **bool** following usual C rules
  - values other than **0** or **1** are not often used here
- Otherwise, treated as a variable name (or possibly as a string) -> the fall-through case

The fall-through case

```
if(<variable|string>)
```

- Expression is either
  - An *unquoted name of a* (possibly undefined) *variable*
  - A *quoted string*

The fall-through case

```
if(<variable|string>)
```

UNQUOTED VARIABLE NAME

- An undefined variable -> **`false`**
  - evaluates to an empty string which is one of the cases of false constants
- The variable's value matches false constants -> **`false`**
- `true` otherwise

```
# Common pattern, often used with variables defined
# by commands such as option(enableSomething "...")
if(enableSomething)
  # ...
endif()
```

The fall-through case

```
if(<variable|string>)
```

QUOTED STRING

- A quoted string that doesn't match any of the defined **`true`** constants always evaluates to `false`

  (CMake 3.1)

The fall-through case

```
if(<variable|string>)
```

QUOTED STRING

- A quoted string that doesn't match any of the defined **`true`** constants always evaluates to `false`

  (CMake 3.1)

Before CMake 3.1, if the value of the string matched the name of an existing variable, then the *quoted string is effectively replaced by that variable name* (unquoted) and the test is then repeated

Another Motivating Example

What will be printed during the configuration phase?

```
cmake_minimum_required(VERSION 3.19)
project(variables NONE)

message(${BUILD_DOCS})

set(BUILD_DOCS ON)
message(${BUILD_DOCS})

set(BUILD_DOCS OFF CACHE BOOL "Docs generation")
message(${BUILD_DOCS})

set(BUILD_DOCS ON)
message(${BUILD_DOCS})
```

Another Motivating Example

What will be printed during the configuration phase?

```
cmake_minimum_required(VERSION 3.19)
project(variables NONE)

message(${BUILD_DOCS})

set(BUILD_DOCS ON)
message(${BUILD_DOCS})

set(BUILD_DOCS OFF CACHE BOOL "Docs generation")
message(${BUILD_DOCS})

set(BUILD_DOCS ON)
message(${BUILD_DOCS})
```

FIRST RUN

```
<empty>
ON
OFF
ON
```

Another Motivating Example

What will be printed during the configuration phase?

```
cmake_minimum_required(VERSION 3.19)
project(variables NONE)

message(${BUILD_DOCS})

set(BUILD_DOCS ON)
message(${BUILD_DOCS})

set(BUILD_DOCS OFF CACHE BOOL "Docs generation")
message(${BUILD_DOCS})

set(BUILD_DOCS ON)
message(${BUILD_DOCS})
```

FIRST RUN

```
<empty>
ON
OFF
ON
```

NEXT RUNS

```
OFF
ON
ON
ON
```

Another Motivating Example

What will be printed during the configuration phase?

```
cmake_minimum_required(VERSION 3.19)
project(variables NONE)

message(${BUILD_DOCS})

set(BUILD_DOCS ON)
message(${BUILD_DOCS})

option(BUILD_DOCS "Docs generation" OFF)
message(${BUILD_DOCS})

set(BUILD_DOCS ON)
message(${BUILD_DOCS})
```

Another Motivating Example

What will be printed during the configuration phase?

```
cmake_minimum_required(VERSION 3.19)
project(variables NONE)

message(${BUILD_DOCS})

set(BUILD_DOCS ON)
message(${BUILD_DOCS})

option(BUILD_DOCS "Docs generation" OFF)
message(${BUILD_DOCS})

set(BUILD_DOCS ON)
message(${BUILD_DOCS})
```

FIRST RUN

```
<empty>
ON
ON
ON
```

Another Motivating Example

What will be printed during the configuration phase?

```
cmake_minimum_required(VERSION 3.19)
project(variables NONE)

message(${BUILD_DOCS})

set(BUILD_DOCS ON)
message(${BUILD_DOCS})

option(BUILD_DOCS "Docs generation" OFF)
message(${BUILD_DOCS})

set(BUILD_DOCS ON)
message(${BUILD_DOCS})
```

FIRST RUN

```
<empty>
ON
ON
ON
```

NEXT RUNS

```
<empty>
ON
ON
ON
```

Another Motivating Example

What will be printed during the configuration phase?

```
cmake_minimum_required(VERSION 3.12)
project(variables NONE)

message(${BUILD_DOCS})

set(BUILD_DOCS ON)
message(${BUILD_DOCS})

option(BUILD_DOCS "Docs generation" OFF)
message(${BUILD_DOCS})

set(BUILD_DOCS ON)
message(${BUILD_DOCS})
```

Another Motivating Example

What will be printed during the configuration phase?

```
cmake_minimum_required(VERSION 3.12)
project(variables NONE)

message(${BUILD_DOCS})

set(BUILD_DOCS ON)
message(${BUILD_DOCS})

option(BUILD_DOCS "Docs generation" OFF)
message(${BUILD_DOCS})

set(BUILD_DOCS ON)
message(${BUILD_DOCS})
```

FIRST RUN

```
<empty>
ON
OFF
ON
```

Another Motivating Example

What will be printed during the configuration phase?

```
cmake_minimum_required(VERSION 3.12)
project(variables NONE)

message(${BUILD_DOCS})

set(BUILD_DOCS ON)
message(${BUILD_DOCS})

option(BUILD_DOCS "Docs generation" OFF)
message(${BUILD_DOCS})

set(BUILD_DOCS ON)
message(${BUILD_DOCS})
```

FIRST RUN

```
<empty>
ON
OFF
ON
```

NEXT RUNS

```
OFF
ON
ON
ON
```

CMake variables gotchas

Normal and cache variables are two separate things. **It is possible to have a normal variable and a cache variable with the same name but holding different values.**

CMake variables gotchas

Normal and cache variables are two separate things. **It is possible to have a normal variable and a cache variable with the same name but holding different values.**

- **Normal variables take precedence over cache variables**
  - `${myVar}` *retrieves normal rather than the cache* variable's value when both have *the same name*
- Setting a *cache variable's value removes a normal variable* of the same name from the current scope if
  - the cache variable **did not exist before** the call to **`set()`**
  - the cache variable **existed before the call** to **`set()`**, but it **did not have a defined type**
  - the **`FORCE` or `INTERNAL` option** was used in the call to **`set()`**

CMake variables gotchas

- Until CMake 3.13 the **`option()`** command behaved the same as the **`set()`** command
  - *normal variables can be removed if a new cache variable is created* or an existing cache variable has an undefined type
- In CMake 3.13 the **behavior of `option()` was changed**
  - if a *normal variable already exists* with the same name, **the command does nothing**

CMake is a great build system generator!

CMake is a great build system generator!

... but a poor general-purpose scripting language.

Recommended Practices

The less CMake the better.

Recommended Practices

The less CMake the better.

- Put to the CMake files only what is really necessary to make the build succeed

Recommended Practices

The less CMake the better.

- **Put to the CMake files only what is really necessary to make the build succeed**
- Choose alternatives when available
  - *dedicated package managers* (Conan/VcPkg) to handle dependencies and packaging of your library
  - *scripting languages* like Python *for CI-like scripts*

Recommended Practices

The less CMake the better.

- **Put to the CMake files only what is really necessary to make the build succeed**
- Choose alternatives when available
  - *dedicated package managers* (Conan/VcPkg) to handle dependencies and packaging of your library
  - *scripting languages* like Python *for CI-like scripts*

CMake **should not** be treated as a general-purpose programming language.

SELECTED FEATURES THAT MADE ME HAPPIER :-)

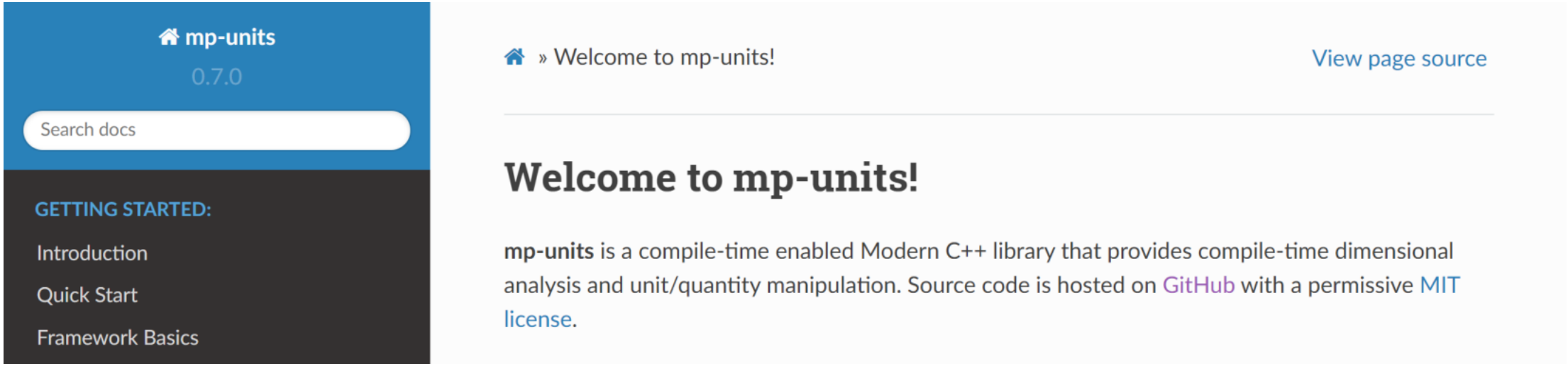
CMAKE

C++20 support (CMake 3.12)

```
cmake_minimum_required(VERSION 3.12)

add_library(mp-units-core INTERFACE)
target_compile_features(mp-units-core INTERFACE cxx_std_20)
```

C++20 support (CMake 3.12)

```
cmake_minimum_required(VERSION 3.12)

add_library(mp-units-core INTERFACE)
target_compile_features(mp-units-core INTERFACE cxx_std_20)
```



Simplified install destination handling (CMake 3.14)

BEFORE

```
install(TARGETS MyLib EXPORT MyLibTargets
  LIBRARY DESTINATION lib
  ARCHIVE DESTINATION lib
  RUNTIME DESTINATION bin
  INCLUDES DESTINATION include
)
```

- Every project *defines* all the *destinations by itself*
- *Poor consistency* among projects
- *Hard to make it correct* for every platform
  - various platforms have various expectations about the layout

Simplified install destination handling (CMake 3.14)

BEFORE

```
install(TARGETS MyLib EXPORT MyLibTargets
  LIBRARY DESTINATION lib
  ARCHIVE DESTINATION lib
  RUNTIME DESTINATION bin
  INCLUDES DESTINATION include
)
```

- Every project *defines* all the *destinations by itself*
- *Poor consistency* among projects
- *Hard to make it correct* for every platform
  - various platforms have various expectations about the layout

AFTER

```
include(GNUInstallDirs)
install(TARGETS MyLib EXPORT MyLibTargets)
```

- Just use the defaults
  - **`GNUInstallDirs`** used if included
  - built-in defaults otherwise

MSVC Compilation Warnings Handling (CMake 3.15)

BEFORE

```
function(set_warnings)
  string(REGEX
    REPLACE "/W[0-4]" "" CMAKE_CXX_FLAGS
    "${CMAKE_CXX_FLAGS}"
  )
  set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS}" PARENT_SCOPE)
  add_compile_options(
    /W4
    # ...
  )
endfunction()
```

- The default value of **CMAKE_<LANG>_FLAGS** for MSVC compilers contains warning level flags (i.e. **/W3**)

MSVC Compilation Warnings Handling (CMake 3.15)

BEFORE

```
function(set_warnings)
  string(REGEX
    REPLACE "/W[0-4]" "" CMAKE_CXX_FLAGS
    "${CMAKE_CXX_FLAGS}"
  )
  set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS}" PARENT_SCOPE)
  add_compile_options(
    /W4
    # ...
  )
endfunction()
```

- The default value of **CMAKE_<LANG>_FLAGS** for MSVC compilers contains warning level flags (i.e. **/W3**)

AFTER

```
function(set_warnings)
  add_compile_options(
    /W4
    # ...
  )
endfunction()
```

- No default warning flags anymore for MSVC

Ninja build

- Probably the best generator
- The broadest platform support
- Very efficient build

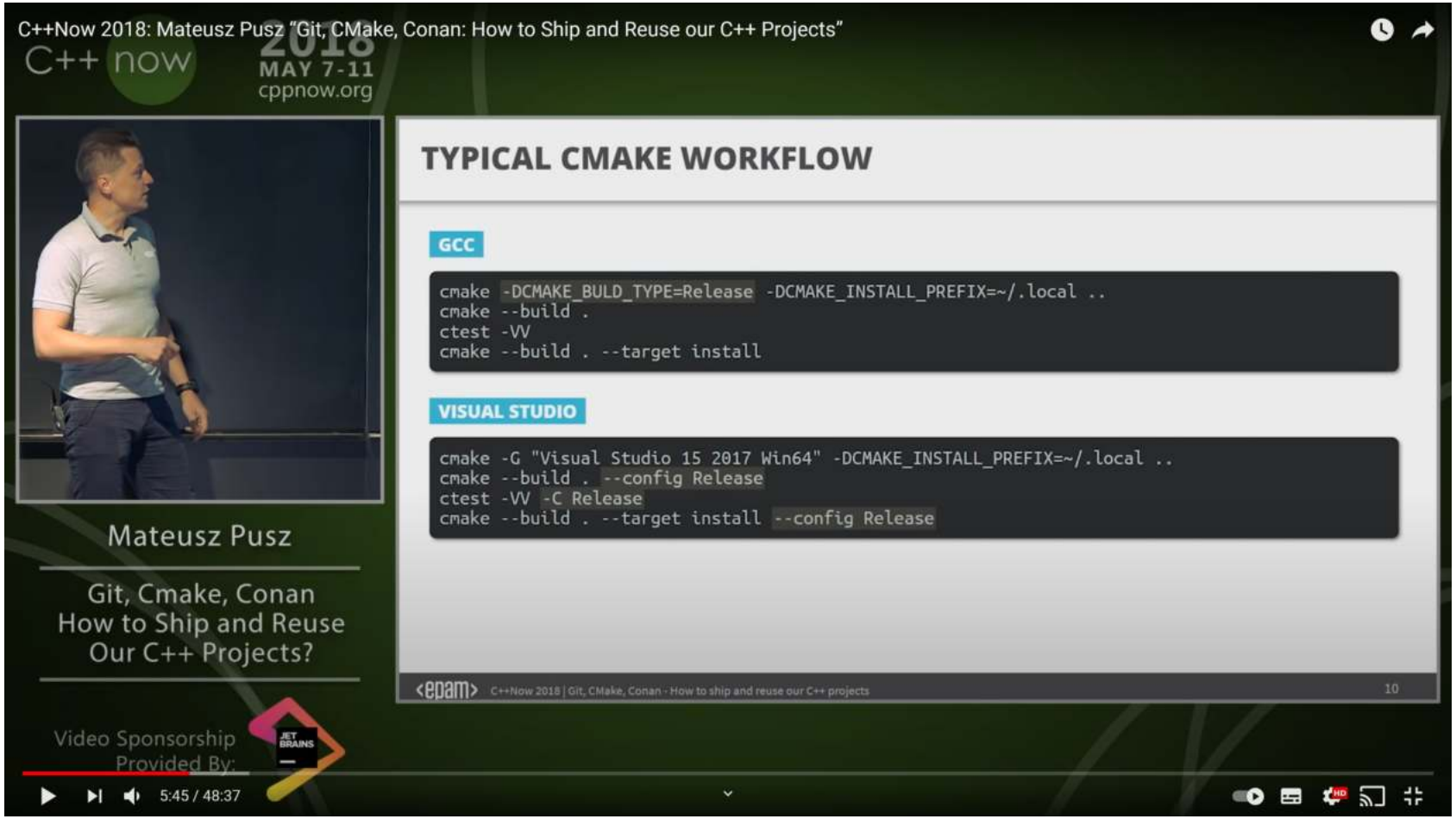
SINGLE CONFIGURATION

```
cmake -G Ninja ...
```

MULTI-CONFIGURATION (CMAKE 3.17)

```
cmake -G "Ninja Multi-Config" ...
```

Typical CMake Workflow (3 years ago)



Executing An Install (CMake 3.15)

- Support for *invoking an install directly* without going through the build tool
  - build step can sometimes be totally omitted (e.g. header-only libraries)

Executing An Install (CMake 3.15)

- Support for *invoking an install directly* without going through the build tool
  - build step can sometimes be totally omitted (e.g. header-only libraries)
- **`cmake --install <bin_dir>`** option
  - **`<bin_dir>`** - the top of the build directory for the CMake project to install

Executing An Install (CMake 3.15)

- Support for *invoking an install directly* without going through the build tool
  - build step can sometimes be totally omitted (e.g. header-only libraries)
- **`cmake --install <bin_dir>`** option
  - **`<bin_dir>`** - the top of the build directory for the CMake project to install
- Followed by additional optional arguments
  - **`--prefix <prefix>`** - where to install to (no need to fix it at install time)
  - **`--config <cfg>`** - which build configuration to use
  - **`--component <comp>`** - which components to install
  - **`--strip`** - whether to perform stripping before carrying out the install

Updated CMake Workflow

NINJA

```
cmake .. -G Ninja -DCMAKE_INSTALL_PREFIX=~/.local -DCMAKE_BUILD_TYPE=Release
cmake --build .
ctest -VV
cmake --install . --strip
```

Updated CMake Workflow

NINJA

```
cmake .. -G Ninja -DCMAKE_INSTALL_PREFIX=~/.local -DCMAKE_BUILD_TYPE=Release
cmake --build .
ctest -VV
cmake --install . --strip
```

NINJA MULTI-CONFIG

```
cmake .. -G "Ninja Multi-Config" -DCMAKE_INSTALL_PREFIX=~/.local
cmake --build . --config Release
ctest -VV -C Release
cmake --install . --config Release --strip
```

Generic Build Script

run.sh

```
cmake .. -G $1 -DCMAKE_INSTALL_PREFIX=~/.local -DCMAKE_BUILD_TYPE=$2
cmake --build . --config $2
ctest -VV -C $2
cmake --install . --config $2 --strip
```

```
./run.sh "Ninja Multi-Config" Release
```

- *Single-configuration* generators *ignore any build-time* specification
- *Multi-configuration* generators *ignore the* ***CMAKE_BUILD_TYPE*** variable

Setting a default generator (CMake 3.15)

.bashrc

```
# ...

# set a default CMake generator
export CMAKE_GENERATOR="Ninja Multi-Config"
```

Setting a default Ninja build type (CMake 3.17)

`.bashrc`

```
# ...

# set a default Ninja Multi-Config build type
export CMAKE_DEFAULT_BUILD_TYPE=Release
```

Works only for a Ninja Multi-Config generator!

Simplified CMake workflow

RELEASE

```
cmake .. -DCMAKE_INSTALL_PREFIX=~/.local
cmake --build .
ctest -VV -C Release
cmake --install . --strip
```

Simplified CMake workflow

RELEASE

```
cmake .. -DCMAKE_INSTALL_PREFIX=~/.local
cmake --build .
ctest -VV -C Release
cmake --install . --strip
```

DEBUG

```
cmake .. -DCMAKE_INSTALL_PREFIX=~/.local
cmake --build . --config Debug
ctest -VV -C Debug
# cmake --install . --config Debug
```

Verbose builds (CMake 3.14)

BEFORE

- Switch generator from Ninja to Makefile
- Enable verbosity for a Makefile generator

```
set(CMAKE_VERBOSE_MAKEFILE ON)
```

Verbose builds (CMake 3.14)

BEFORE

- Switch generator from Ninja to Makefile
- Enable verbosity for a Makefile generator

```
set(CMAKE_VERBOSE_MAKEFILE ON)
```

NOW

- Enable verbosity with a build step command line **`-v|--verbose`** flag

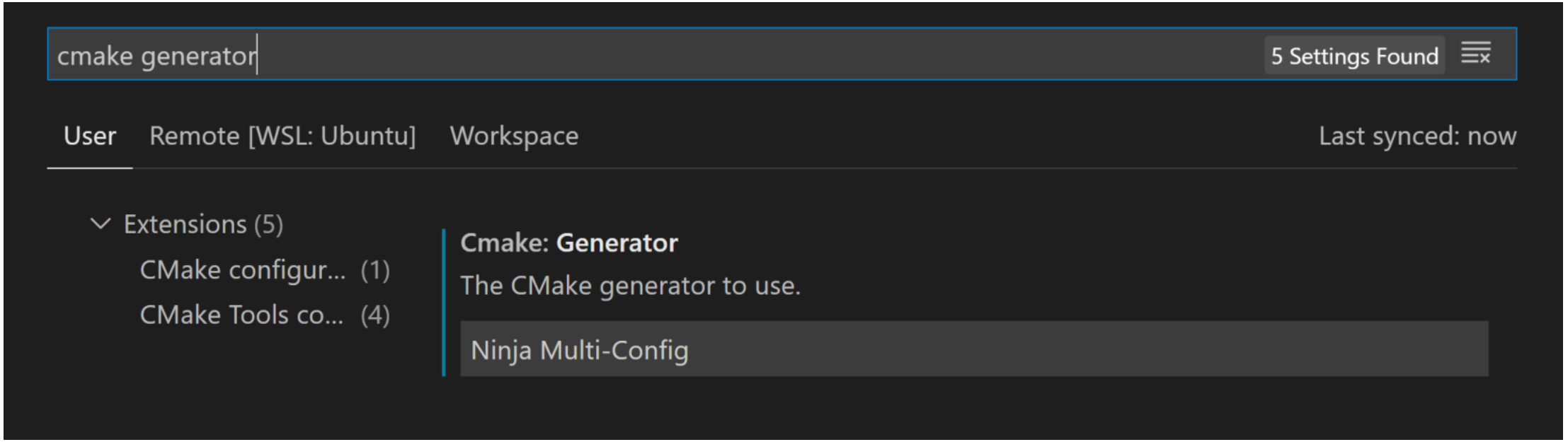
```
cmake --build . -v
```

File-Based API (CMake 3.14)

- *Improves integration* of CMake *with IDEs*
  - supported by every major IDE (Visual Studio, VSCode, CLion, QtCreator, ...)
- **Better performance**
  - no need to *run CMake server mode* or *parse files* generated by the CMake configure stage
- **Less memory usage**

File-Based API (CMake 3.14)

- Allows the usage of any CMake generator within the IDE



Preferring user-provided packages (CMake 3.15)

BEFORE

```
find_package(GTest CONFIG REQUIRED)

add_executable(unit-tests
  # ...
)
target_link_libraries(unit-tests PRIVATE
  GTest::gtest_main
)
```

- Most modern CMake projects *generate config files* on installation
- **`find_package()`** without **`CONFIG`** *prefers a system installed library* over the one provided by the user

Preferring user-provided packages (CMake 3.15)

BEFORE

```
find_package(GTest CONFIG REQUIRED)

add_executable(unit-tests
  # ...
)
target_link_libraries(unit-tests PRIVATE
  GTest::gtest_main
)
```

- Most modern CMake projects *generate config files* on installation
- **`find_package()`** without **`CONFIG`** *prefers a system installed library* over the one provided by the user

AFTER

```
set(CMAKE_FIND_PACKAGE_PREFER_CONFIG ON)
```

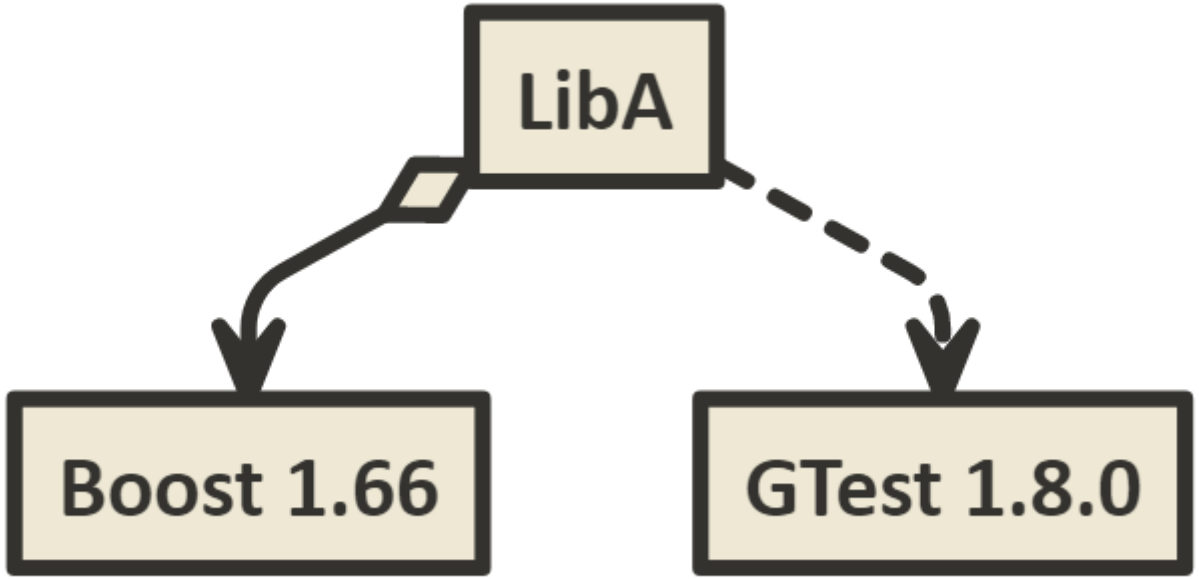
```
find_package(GTest REQUIRED)

add_executable(unit-tests
  # ...
)
target_link_libraries(unit-tests PRIVATE
  GTest::gtest_main
)
```

- Makes **`find_package()`** calls to look for a *package configuration file first* even if a find module is available
- Simplifies the usage of Package Managers

MODERN PROJECT STRUCTURE

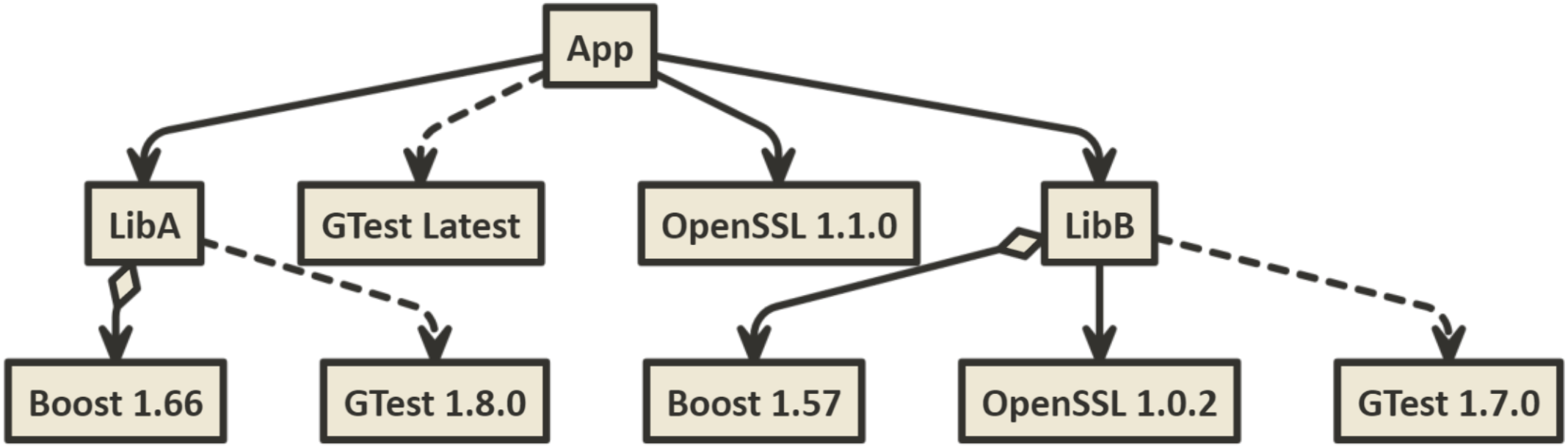
REFRESH

`add_subdirectory()` for deps?

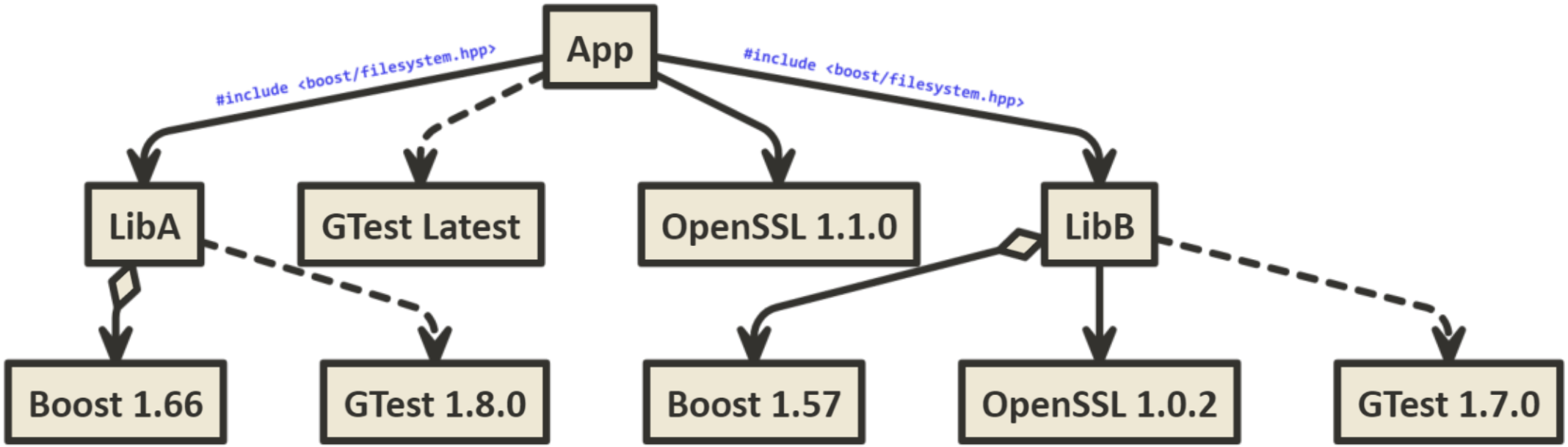


`add_subdirectory()` for deps?

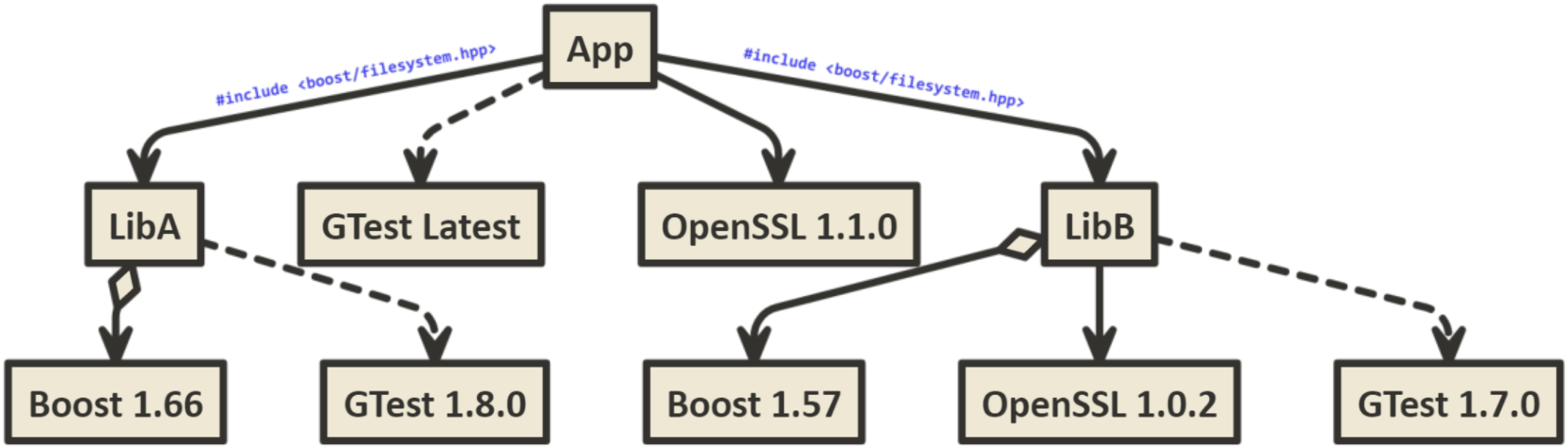
`add_subdirectory()` for deps?



add_subdirectory() for deps? Please DON'T!



`add_subdirectory()` for deps? Please DON'T!



Handling dependencies as subdirectories **does not scale**!

Not `IMPORTED` CMake targets have a global scope!

Even if there are *no version* conflicts, **`add_subdirectory`** still **does not scale**.

Not `IMPORTED` CMake targets have a global scope!

Even if there are *no version* conflicts, **`add_subdirectory`** still **does not scale**.

- Combining multiple projects into a single build via **`add_subdirectory()`** *may lead to name collisions*
  - targets will be *duplicated* if the same dependency is *added more than once*
  - different projects may define *targets with the same name* (**`core`**, **`utils`**, **`unit_tests`**, ...)
- CMake requires all **global targets to be unique**

Not `IMPORTED` CMake targets have a global scope!

Even if there are *no version* conflicts, **`add_subdirectory`** still **does not scale**.

- Combining multiple projects into a single build via **`add_subdirectory()`** *may lead to name collisions*
  - targets will be *duplicated* if the same dependency is *added more than once*
  - different projects may define *targets with the same name* (**`core`**, **`utils`**, **`unit_tests`**, ...)
- CMake requires all **global targets to be unique**

The more projects you have in your company the more probability that your target names will collide.

Recommended Practices

It is a good practice to always prefix every public CMake target's name with the name of the project.

```
add_library(myproj-core
  source_1.cpp
  source_2.cpp
  # ...
)
add_library(myproj::core ALIAS myproj-core)
```

Recommended Practices

It is a good practice to always prefix every public CMake target's name with the name of the project.

```
add_library(myproj-core
  source_1.cpp
  source_2.cpp
  # ...
)
add_library(myproj::core ALIAS myproj-core)
```

Even though you should not use **`add_subdirectory()`** for dependencies, you can make it easier for those that will choose otherwise.

Avoid repeating the prefix in the exported name

```
add_library(myproj-core
  source_1.cpp
  source_2.cpp
  # ...
)
set_target_properties(myproj-core PROPERTIES EXPORT_NAME core)
add_library(myproj::core ALIAS myproj-core)

install(TARGETS myproj-core EXPORT MyProjTargets)
install(EXPORT MyProjTargets
  DESTINATION ${CMAKE_INSTALL_LIBDIR}/cmake/myproj
  NAMESPACE myproj::
)
```

The target's **EXPORT_NAME** property can be set to use a different name when exporting the target.

What about project's private targets?

- Projects often *define other CMake targets* that are **not exported by the library**
  - unit tests
  - usage examples
  - documentation generation
  - ...

What about project's private targets?

- Projects often *define other CMake targets* that are **not exported by the library**
  - unit tests
  - usage examples
  - documentation generation
  - ...
- Dependers **should not be forced to add them**
  - pollute the CMake project with *external targets*
  - pollute CTest with *external unit tests*
  - *targets with the same names may collide* between projects

What about project's private targets?

- Projects often *define other CMake targets* that are **not exported by the library**
  - unit tests
  - usage examples
  - documentation generation
  - ...
- Dependers **should not be forced to add them**
  - pollute the CMake project with *external targets*
  - pollute CTest with *external unit tests*
  - *targets with the same names may collide* between projects

Modern Project structure helps a lot here.

Modern Project Structure

Designed to help separate project development workflow from its usage by dependers.

Modern Project Structure

Designed to help separate project development workflow from its usage by dependers.

DEPENDERS SHOULD NOT

- Be forced to *include not exported targets*
  - unit tests
  - usage examples
  - ...
- Be affected by our *development environment*
  - compilation warning flags
  - sanitizers usage
  - ...

Modern Project Structure

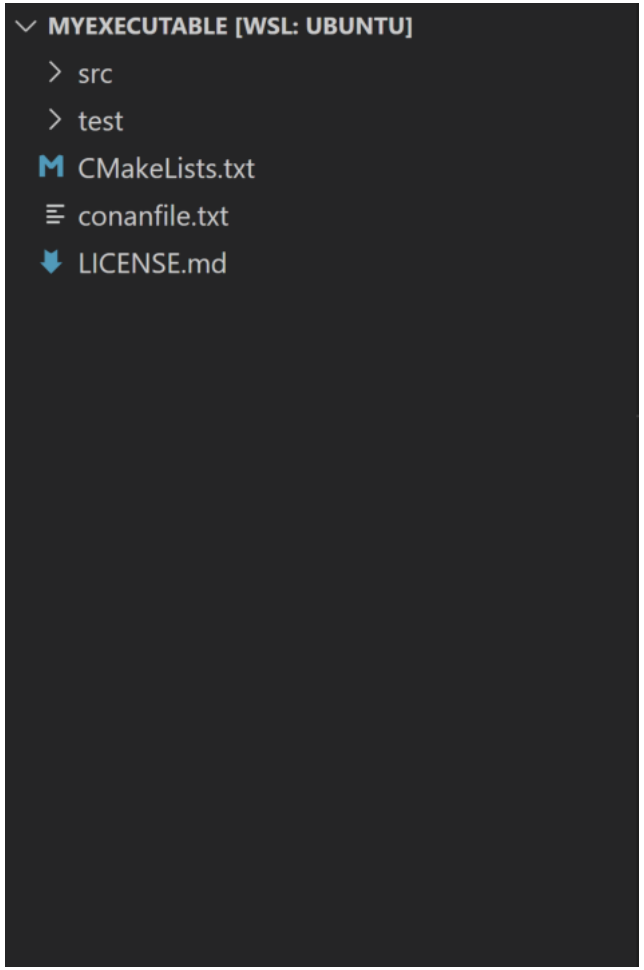
Designed to help separate project development workflow from its usage by dependers.

DEPENDERS SHOULD NOT

- Be forced to *include not exported targets*
  - unit tests
  - usage examples
  - ...
- Be affected by our *development environment*
  - compilation warning flags
  - sanitizers usage
  - ...

DISCLAIMER: This is not the only way to solve the problem. This is my way to solve it. You may either follow it or invent something that works for you.

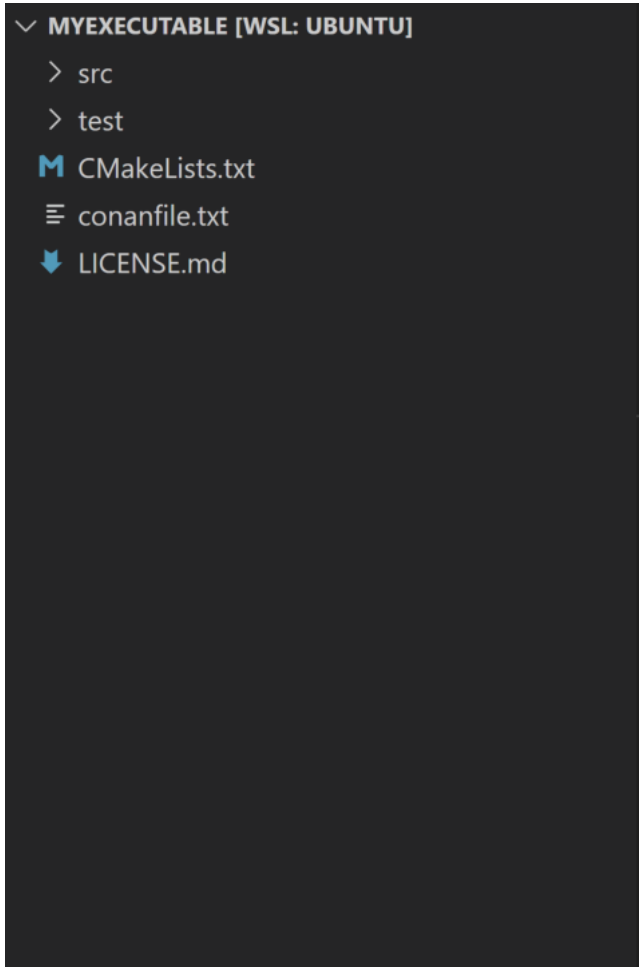
Modern Project Structure: Executable



Modern Project Structure: Executable

`./conanfile.txt`

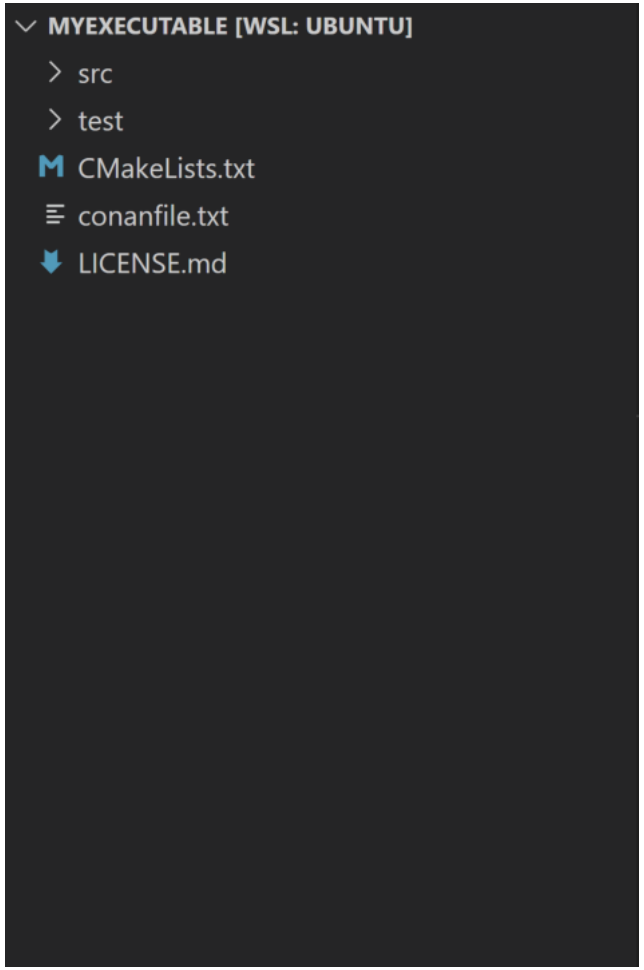
- List of **project's dependencies and their configuration**
- Provides a build system's specific *generator*



Modern Project Structure: Executable

`./CMakeLists.txt`

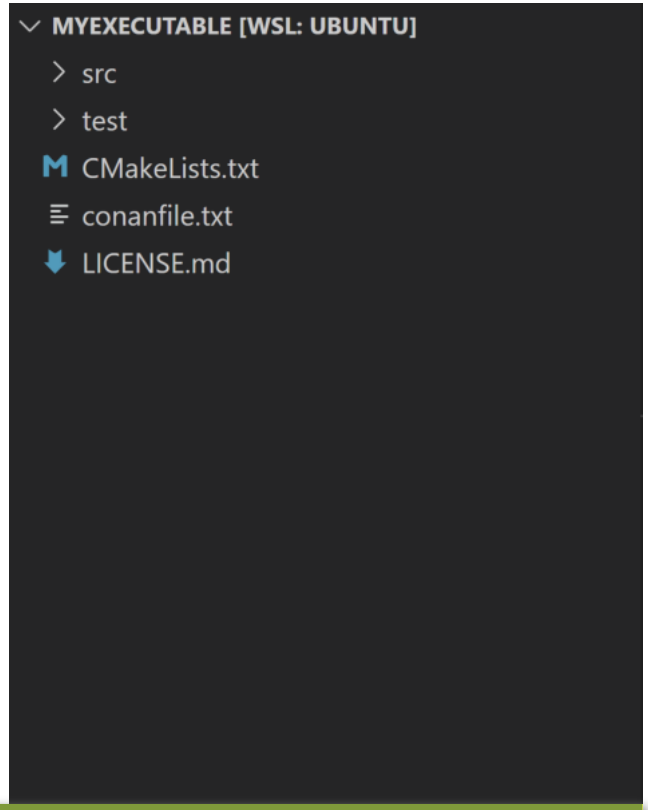
- Entry point for **development**
- Enables all **development-specific features**
  - compilation warnings
  - unit tests
  - building docs
  - ...
- Simple project *wrapper*
  - **`src`** and **`test`** subdirectories added with **`add_subdirectory()`**



Modern Project Structure: Executable

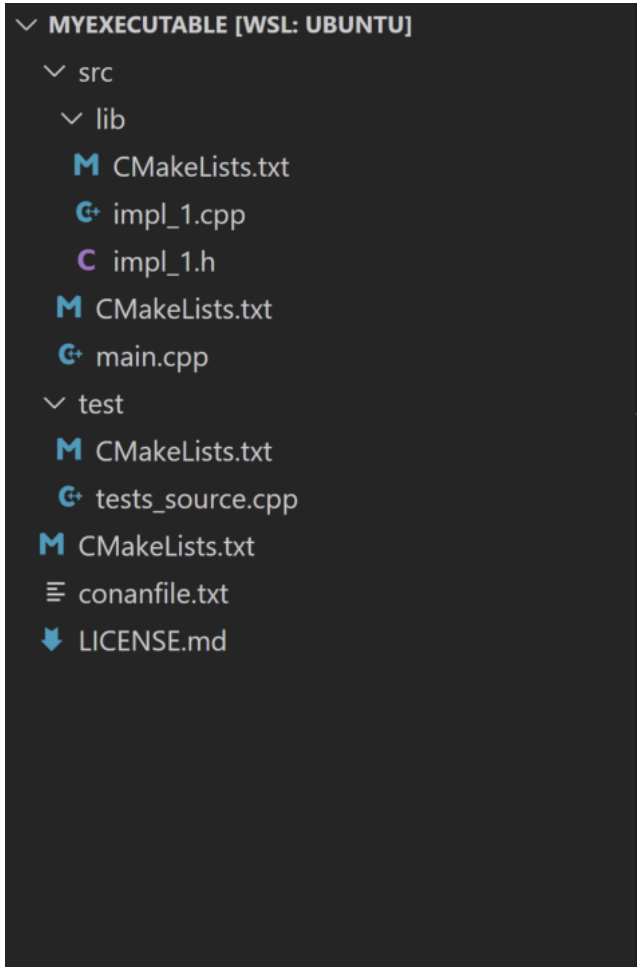
`./CMakeLists.txt`

- Entry point for **development**
- Enables all **development-specific features**
  - compilation warnings
  - unit tests
  - building docs
  - ...
- Simple project *wrapper*
  - **`src`** and **`test`** subdirectories added with **`add_subdirectory()`**



This file should be used by the project developers (i.e. opened in the IDE).

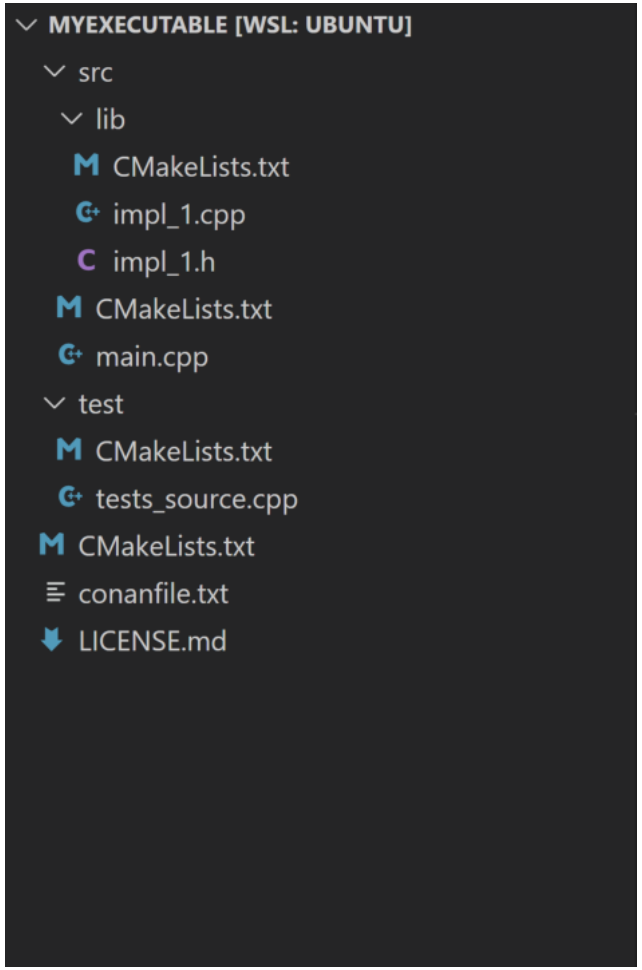
Modern Project Structure: Executable



Modern Project Structure: Executable

`./src/lib/CMakeLists.txt`

- Defines a **library target** with most of the project's implementation
- Used as an *input* for
  - the *executable*
  - *unit tests*

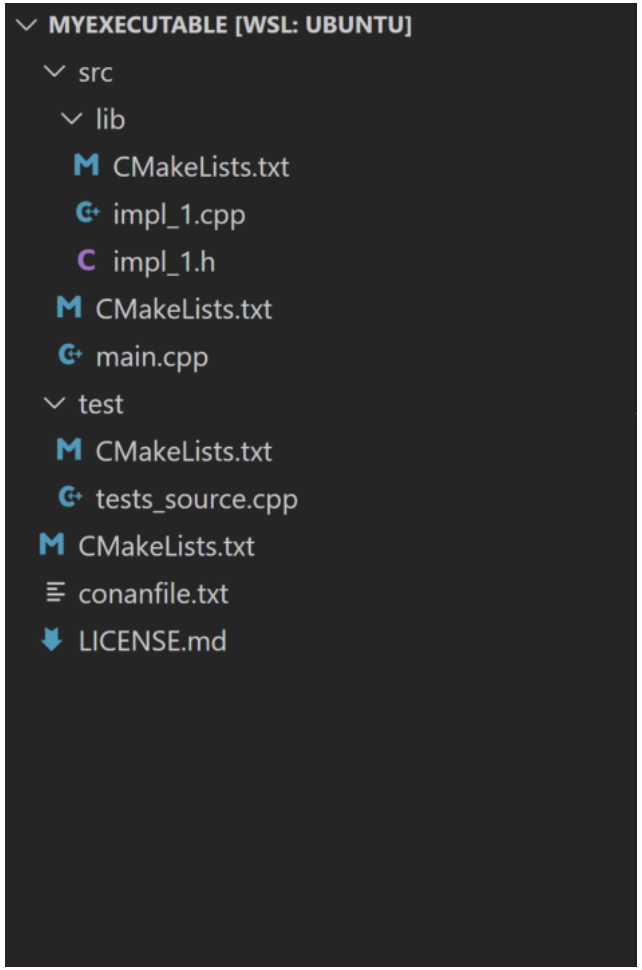


Modern Project Structure: Executable

`./src/CMakeLists.txt`

- Standalone CMake file
- Entry point for customers
- Defines the final executable
- **`lib`** subdirectory added with

  `add_subdirectory()`
- Doesn't change development environment!

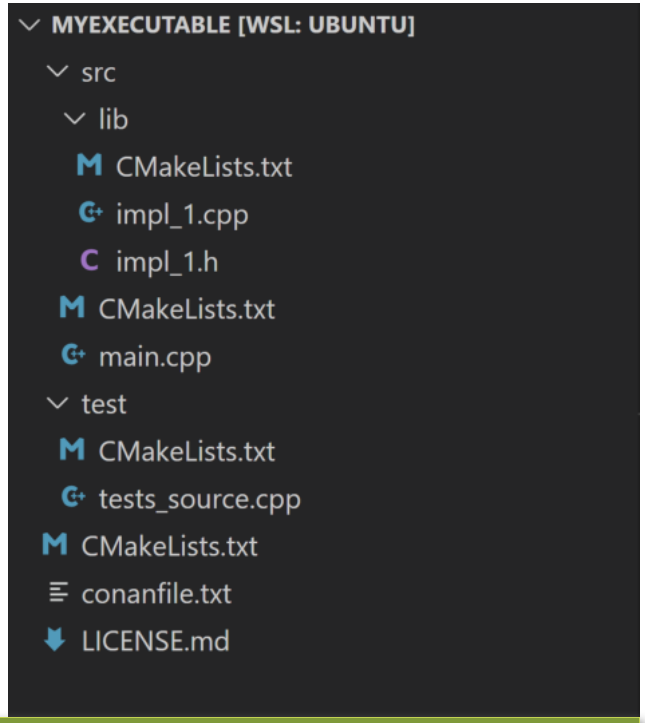


Modern Project Structure: Executable

`./src/CMakeLists.txt`

- Standalone CMake file
- Entry point for customers
- Defines the final executable
- **`lib`** subdirectory added with

  `add_subdirectory()`
- Doesn't change development environment!



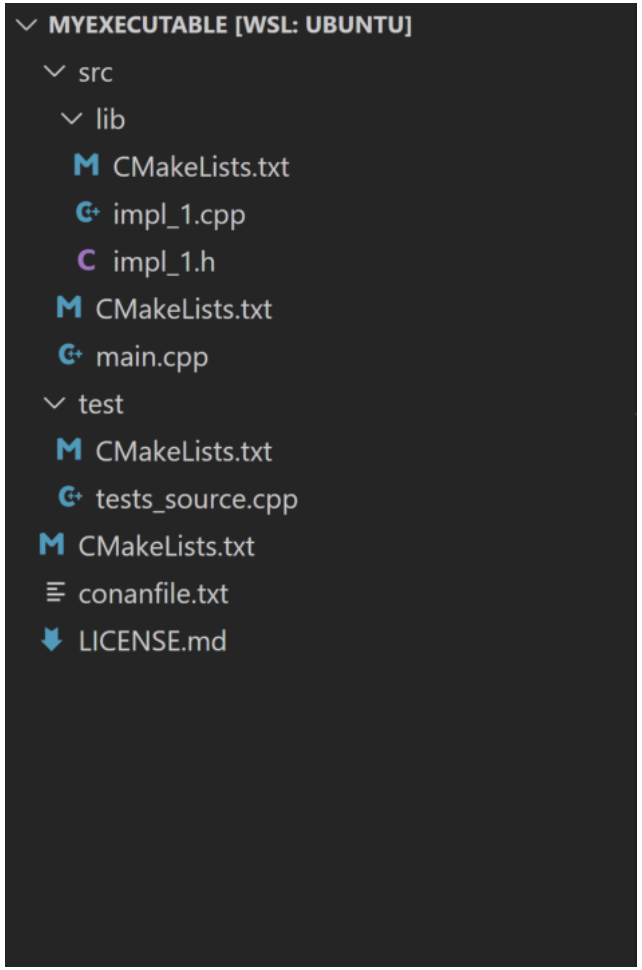
This file should be used by users that need to build the executable but do not want to run unit tests or set compilation warnings used by the DEV team.

Modern Project Structure: Executable

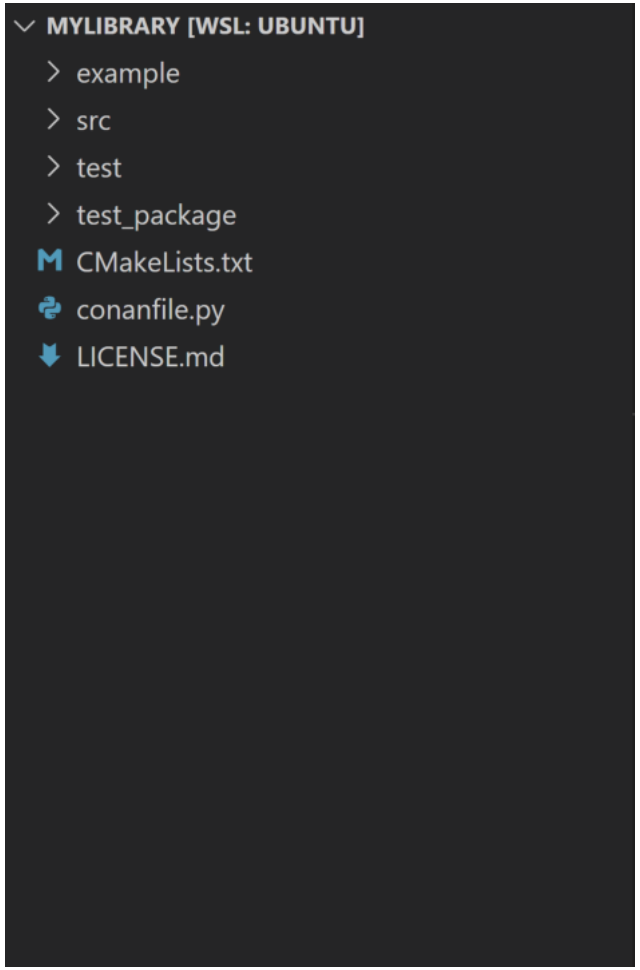
`./test/CMakeLists.txt`

- Included by the **`./CMakeLists.txt`**
- Defines unit tests executable
- Links with a target defined in

  `./src/lib/CMakeLists.txt`



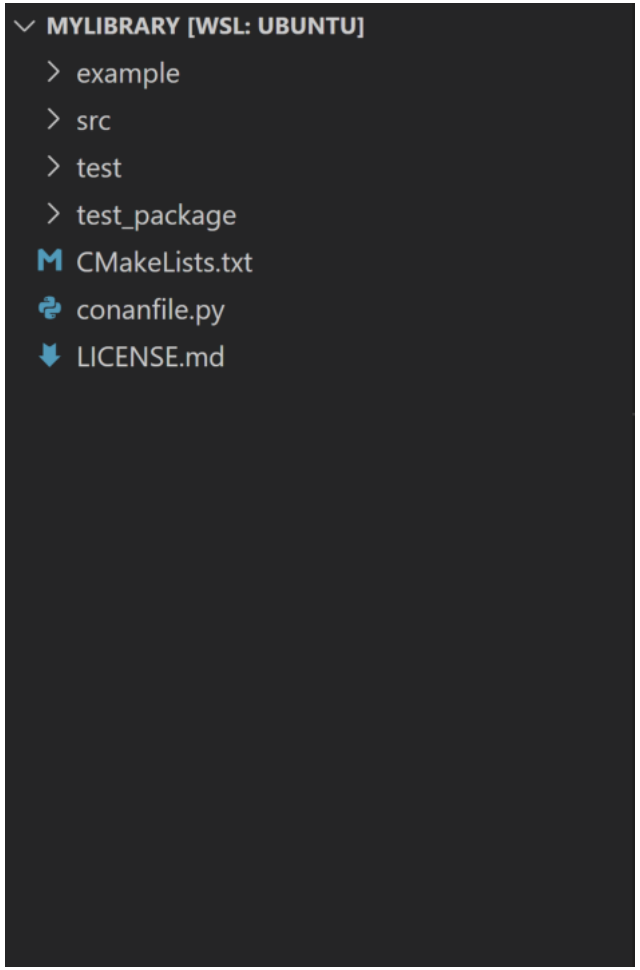
Modern Project Structure: Library



Modern Project Structure: Library

`./conanfile.py`

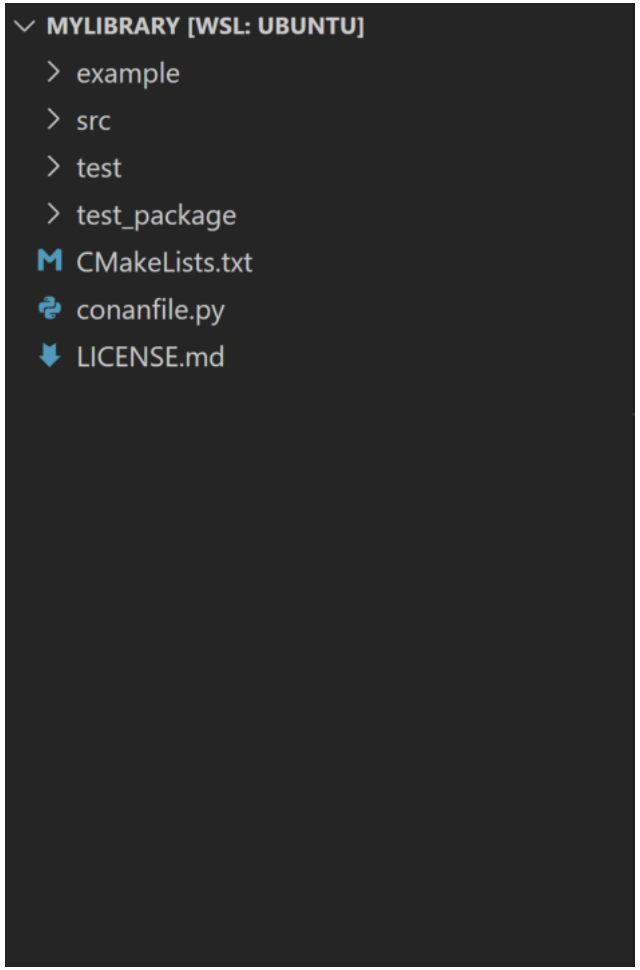
- List of project's dependencies and their configuration
- Provides a full recipe to build, package, and reuse the library with Conan
- Initializes the CMake toolchain



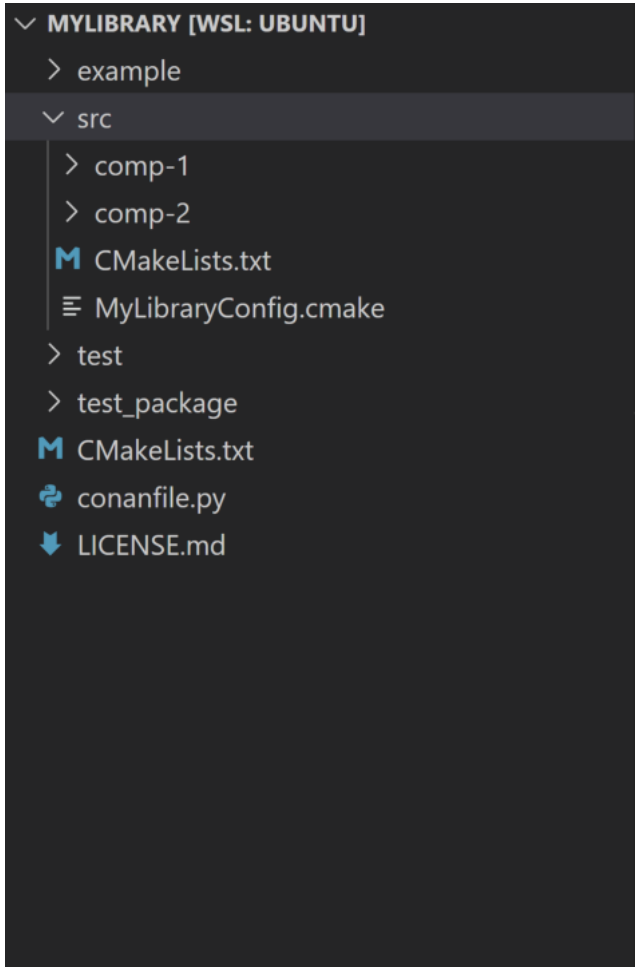
Modern Project Structure: Library

`./CMakeLists.txt`

- Entry point for development
- Enables all development-specific features
  - compilation warnings
  - unit tests
  - usage examples
  - building docs
  - ...
- Simple project *wrapper*
  - **`src`**, **`example`**, and **`test`** subdirectories added with **`add_subdirectory()`**



Modern Project Structure: Library

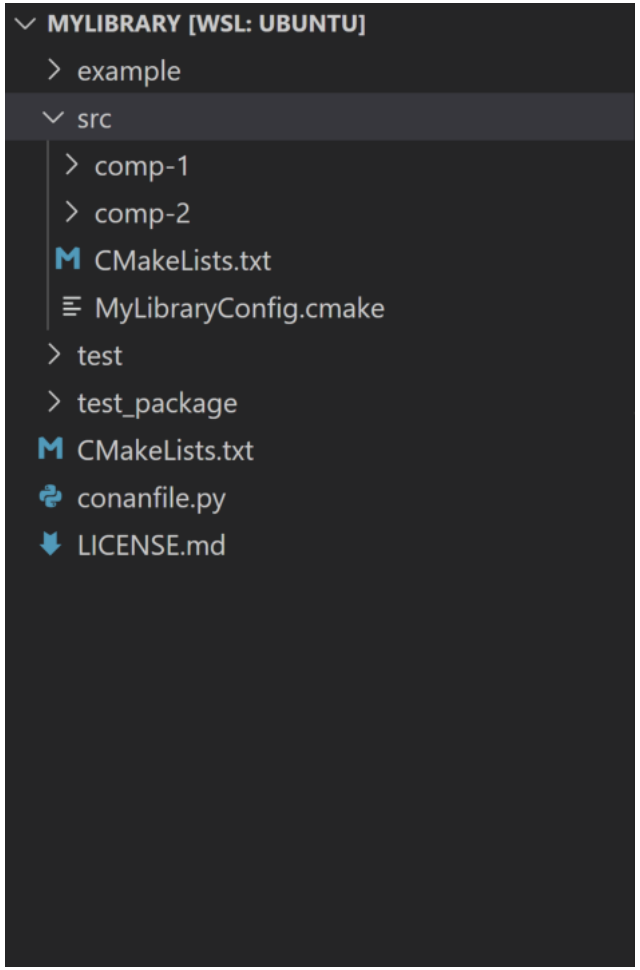


Modern Project Structure: Library

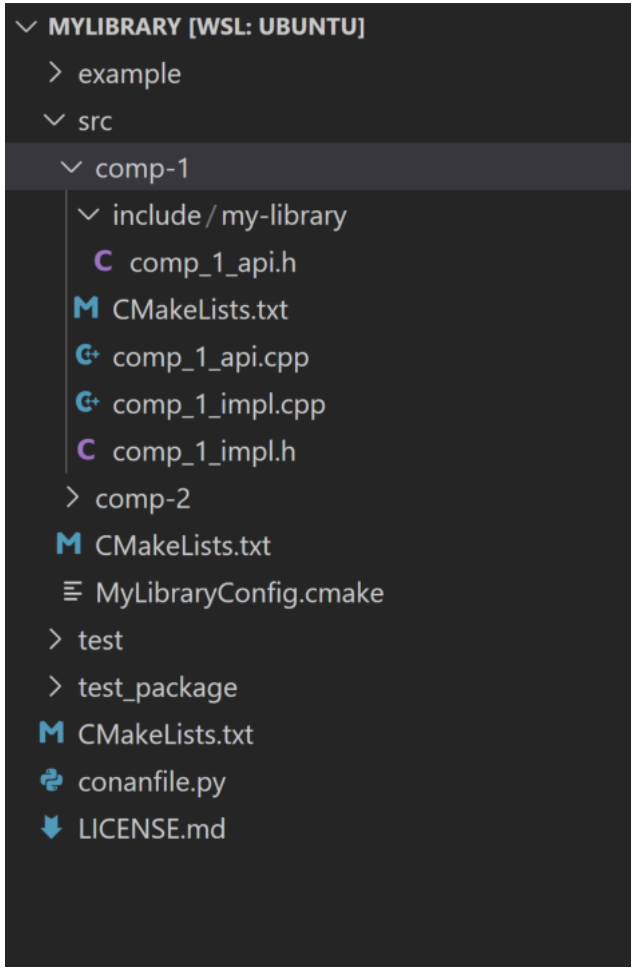
`./src/CMakeLists.txt`

- Standalone CMake file
- Entry point for customers
- Component's subdirectories added with

  `add_subdirectory()`
- Doesn't change development environment!
  - compilation warnings
  - sanitizers
  - ...



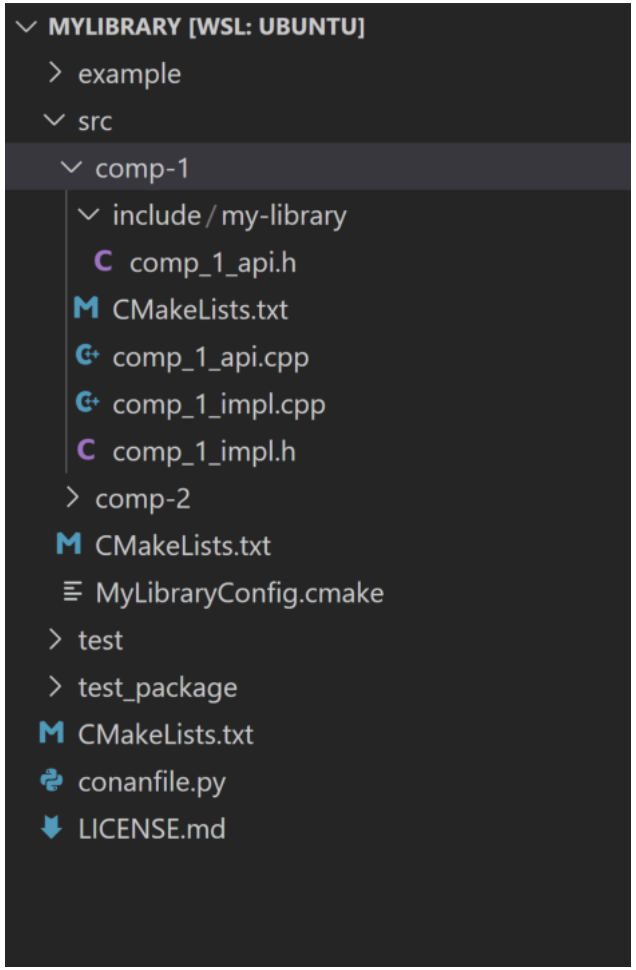
Modern Project Structure: Library



Modern Project Structure: Library

`./src/comp-{N}`

- Implementation of the library's **components**
- Easy to *enable/disable* based on the project's configuration
- Not needed if only one component provided



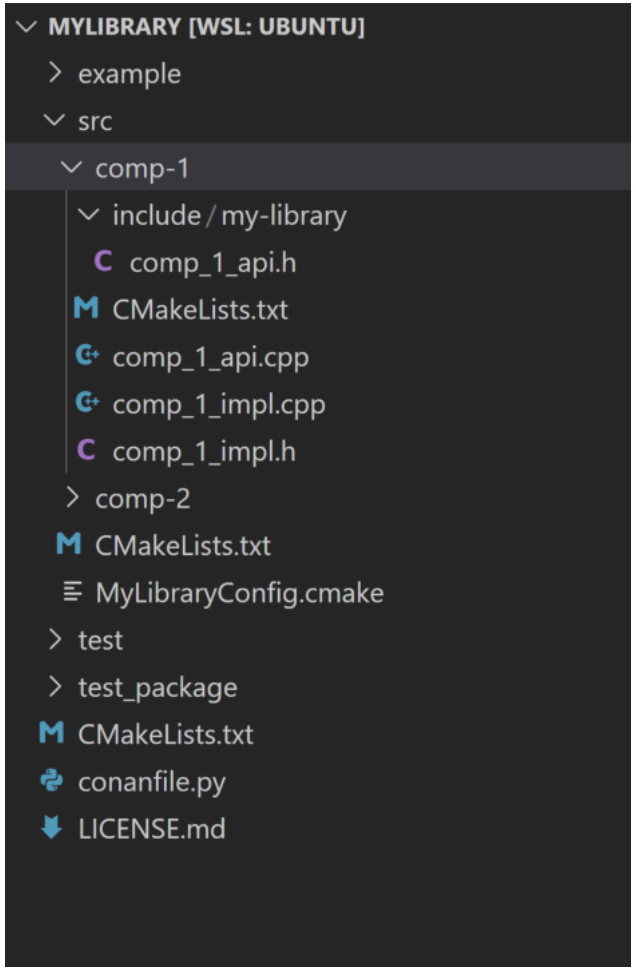
Modern Project Structure: Library

`./src/comp-{N}`

- Implementation of the library's **components**
- Easy to *enable/disable* based on the project's configuration
- Not needed if only one component provided

`./src/comp-{N}/*.h`

- **Private interface** of the component



Modern Project Structure: Library

`./src/comp-{N}`

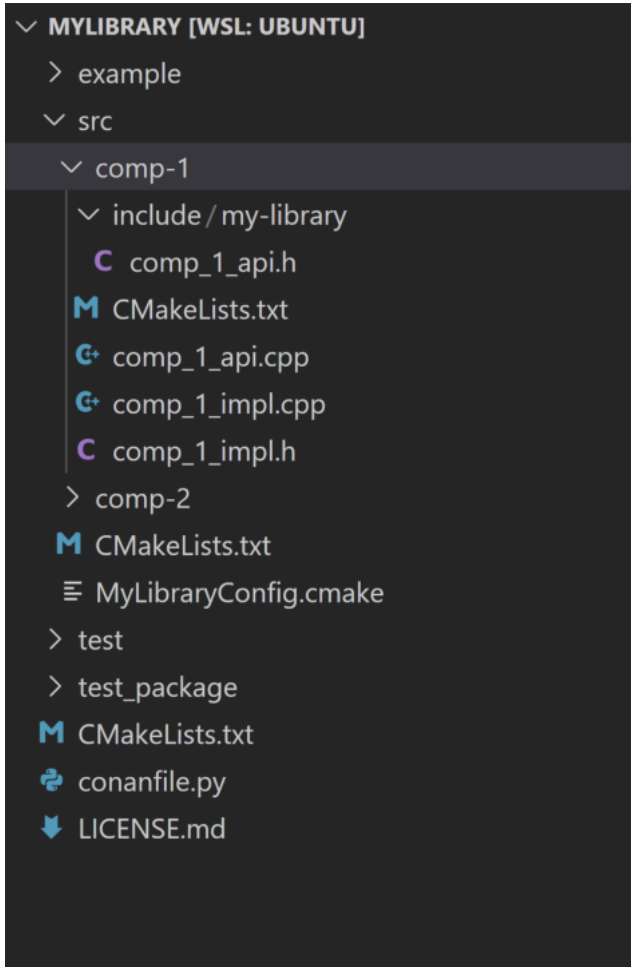
- Implementation of the library's components
- Easy to *enable/disable* based on the project's configuration
- Not needed if only one component provided

`./src/comp-{N}/*.h`

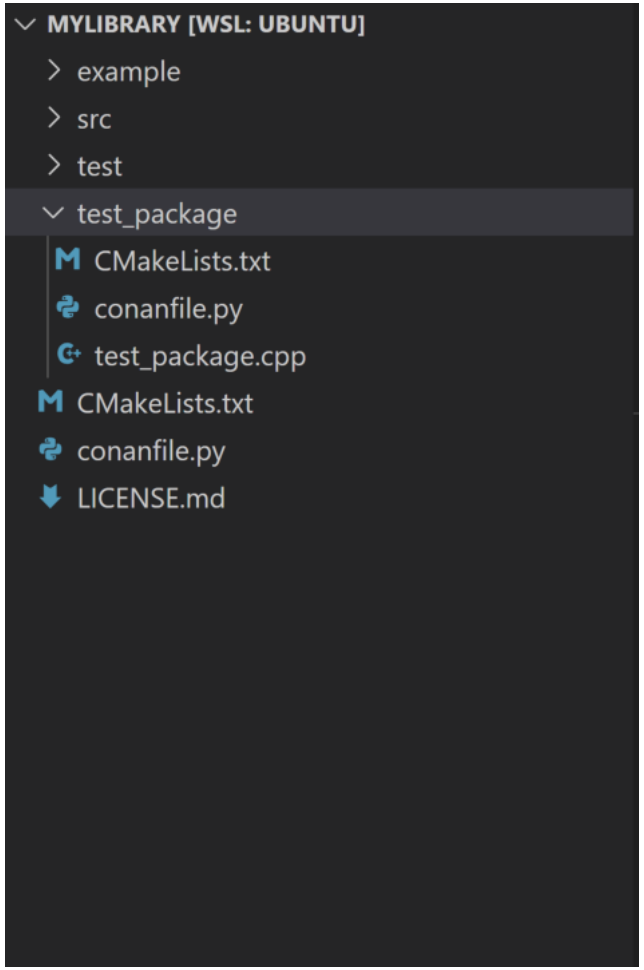
- Private interface of the component

`./src/comp-{N}/include`

- Provides component's public interface
- Entire directory *copied in the installation step*



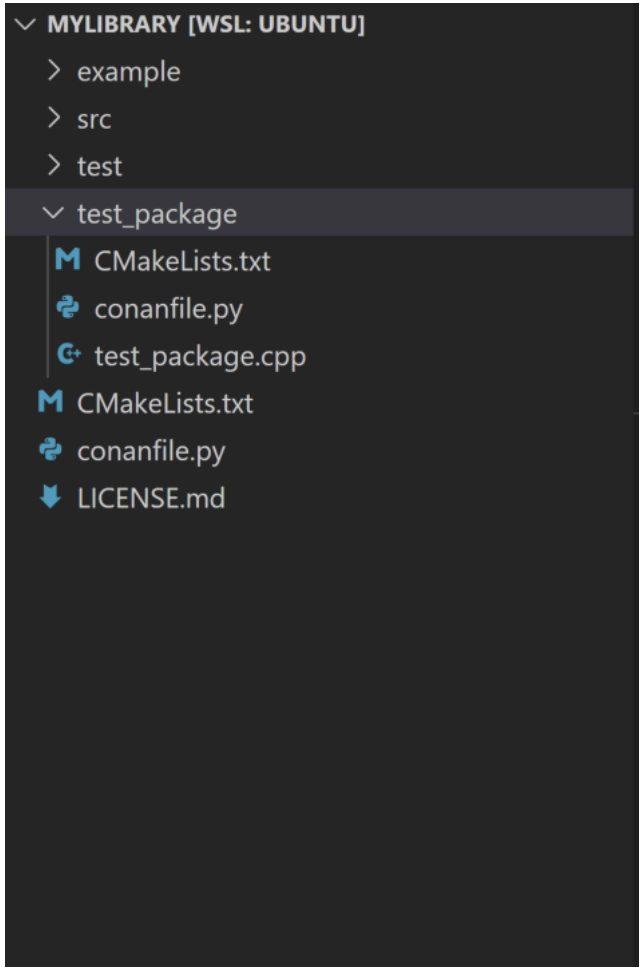
Modern Project Structure: Library



Modern Project Structure: Library

`./test_package/CMakeLists.txt`

- Standalone CMake file
  - not included by other CMake files



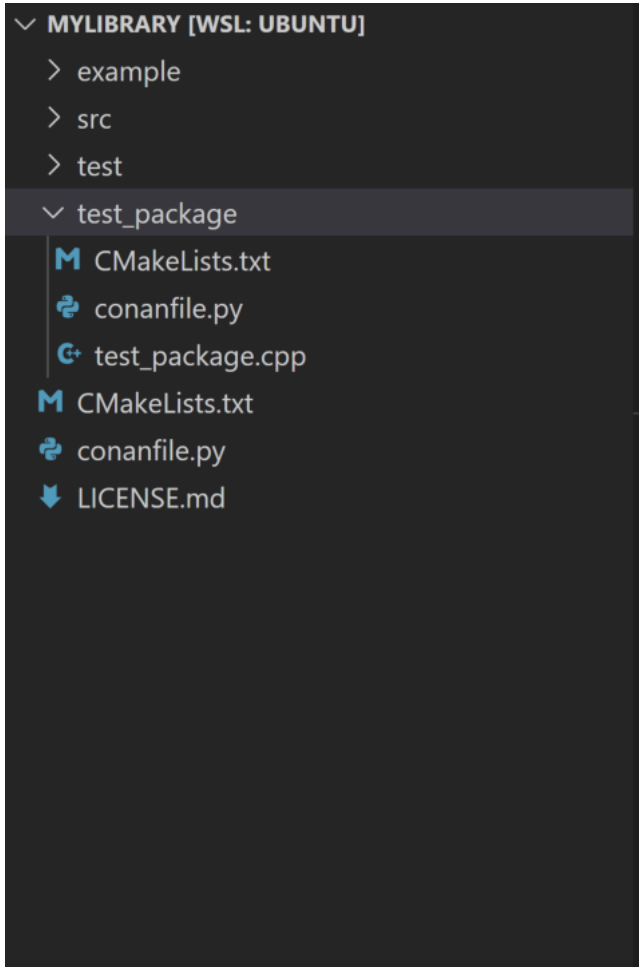
Modern Project Structure: Library

`./test_package/CMakeLists.txt`

- Standalone CMake file
  - not included by other CMake files

`./test_package/test_package.cpp`

- Simple test verifying the packaging process
- Useful to test both Conan and CMake-based installation



Modern Project Structure: Use Cases

Modern Project Structure: Use Cases

PROJECT DEVELOPERS

- Use **`./CMakeLists.txt`** wrapper
- *Open* it in the IDE
- Use from the *command line*

```
mkdir build && cd build
cmake .. -DCMAKE_INSTALL_PREFIX=~/.local
cmake --build .
ctest -VV -C Release
cmake --install . --strip
```

Modern Project Structure: Use Cases

PROJECT DEVELOPERS

- Use **`./CMakeLists.txt`** wrapper
- *Open* it in the IDE
- Use from the *command line*

```
mkdir build && cd build
cmake .. -DCMAKE_INSTALL_PREFIX=~/.local
cmake --build .
ctest -VV -C Release
cmake --install . --strip
```

USERS

- Directly use **`./src/CMakeLists.txt`**
- Users *do not need to open* the dependency's project in the IDE

```
mkdir build && cd build
cmake ../src -DCMAKE_INSTALL_PREFIX=~/.local
cmake --build .
cmake --install . --strip
```

Modern Project Structure: Use Cases

PROJECT DEVELOPERS

- Use **`./CMakeLists.txt`** wrapper
- *Open* it in the IDE
- Use from the *command line*

```
mkdir build && cd build
cmake .. -DCMAKE_INSTALL_PREFIX=~/.local
cmake --build .
ctest -VV -C Release
cmake --install . --strip
```

USERS

- Directly use **`./src/CMakeLists.txt`**
- Users *do not need to open* the dependency's project in the IDE

```
mkdir build && cd build
cmake ../src -DCMAKE_INSTALL_PREFIX=~/.local
cmake --build .
cmake --install . --strip
```

- Typically *no interest in compiling and running project's unit tests*
  - otherwise, use **`./CMakeLists.txt`** if you want to be extra sure that everything is OK before installing

A Need For Installing

Keeping dependencies as subdirectories in a project's file tree is irrational.

A Need For Installing

Keeping dependencies as subdirectories in a project's file tree is irrational.

- Have to be *(re)downloaded each time* for every project

A Need For Installing

Keeping dependencies as subdirectories in a project's file tree is irrational.

- Have to be *(re)downloaded each time* for every project
- Multiple copies of *the same sources take a lot of disk space*

A Need For Installing

Keeping dependencies as subdirectories in a project's file tree is irrational.

- Have to be *(re)downloaded each time* for every project
- Multiple copies of *the same sources take a lot of disk space*
- Have to be *(re)compiled* with every clean build of every project

A Need For Installing

Keeping dependencies as subdirectories in a project's file tree is irrational.

- Have to be *(re)downloaded each time* for every project
- Multiple copies of *the same sources take a lot of disk space*
- Have to be *(re)compiled* with every clean build of every project
- *Build Artifacts take even more space*

A Need For Installing

Keeping dependencies as subdirectories in a project's file tree is irrational.

- Have to be *(re)downloaded each time* for every project
- Multiple copies of *the same sources take a lot of disk space*
- Have to be *(re)compiled* with every clean build of every project
- *Build Artifacts take even more space*
- Pollute the CMake project with *external targets* and CTest with *external unit tests*

A Need For Installing

Keeping dependencies as subdirectories in a project's file tree is irrational.

- Have to be *(re)downloaded each time* for every project
- Multiple copies of *the same sources take a lot of disk space*
- Have to be *(re)compiled* with every clean build of every project
- *Build Artifacts take even more space*
- Pollute the CMake project with *external targets* and CTest with *external unit tests*
- *Targets with the same names may collide* between projects

Installation Process Issues

- **Separate directories needed** for every option that may affect the ABI

Installation Process Issues

- **Separate directories needed** for every option that may affect the ABI
  - *build types* (Debug, Release, MinSizeRel, RelWithDebInfo, ...)

Installation Process Issues

- **Separate directories needed** for every option that may affect the ABI
  - *build types* (Debug, Release, MinSizeRel, RelWithDebInfo, ...)
  - *compilers* (gcc-10, gcc-7.3, clang-11, ...)

Installation Process Issues

- **Separate directories needed** for every option that may affect the ABI
  - *build types* (Debug, Release, MinSizeRel, RelWithDebInfo, ...)
  - *compilers* (gcc-10, gcc-7.3, clang-11, ...)
  - *C++ Standard Library* (libstdc++, libstdc++11, libc++)

Installation Process Issues

- **Separate directories needed** for every option that may affect the ABI
  - *build types* (Debug, Release, MinSizeRel, RelWithDebInfo, ...)
  - *compilers* (gcc-10, gcc-7.3, clang-11, ...)
  - *C++ Standard Library* (libstdc++, libstdc++11, libc++)
  - *C++ Standard Version* (C++11, C++14, C++17, C++20, ...)

Installation Process Issues

- **Separate directories needed** for every option that may affect the ABI
  - *build types* (Debug, Release, MinSizeRel, RelWithDebInfo, ...)
  - *compilers* (gcc-10, gcc-7.3, clang-11, ...)
  - *C++ Standard Library* (libstdc++, libstdc++11, libc++)
  - *C++ Standard Version* (C++11, C++14, C++17, C++20, ...)
  - *Project preprocessor flags*

Installation Process Issues

- **Separate directories needed** for every option that may affect the ABI
  - *build types* (Debug, Release, MinSizeRel, RelWithDebInfo, ...)
  - *compilers* (gcc-10, gcc-7.3, clang-11, ...)
  - *C++ Standard Library* (libstdc++, libstdc++11, libc++)
  - *C++ Standard Version* (C++11, C++14, C++17, C++20, ...)
  - *Project preprocessor flags*
  - ...

Installation Process Issues

- **Separate directories needed** for every option that may affect the ABI
  - *build types* (Debug, Release, MinSizeRel, RelWithDebInfo, ...)
  - *compilers* (gcc-10, gcc-7.3, clang-11, ...)
  - *C++ Standard Library* (libstdc++, libstdc++11, libc++)
  - *C++ Standard Version* (C++11, C++14, C++17, C++20, ...)
  - *Project preprocessor flags*
  - ...
- **Unless the package does not need** the above
  - i.e. header-only library

Installation Process Issues

- **Separate directories needed** for every option that may affect the ABI
  - *build types* (Debug, Release, MinSizeRel, RelWithDebInfo, ...)
  - *compilers* (gcc-10, gcc-7.3, clang-11, ...)
  - *C++ Standard Library* (libstdc++, libstdc++11, libc++)
  - *C++ Standard Version* (C++11, C++14, C++17, C++20, ...)
  - *Project preprocessor flags*
  - ...
- **Unless the package does not need** the above
  - i.e. header-only library

In general, as a user **you never know**, so it is better to be on the safe side.

Installation Process Issues

- Cumbersome, space and time wasting

Installation Process Issues

- **Cumbersome, space and time wasting**
- Most packages are **not ready to install several versions** of the same product at once
  - on install they do not produce *subdirectories for each package version*
  - big pain if *your projects require specific but different versions* of dependencies

Installation Process Issues

- **Cumbersome, space and time wasting**
- Most packages are **not ready to install several versions** of the same product at once
  - on install they do not produce *subdirectories for each package version*
  - big pain if *your projects require specific but different versions* of dependencies
- **What if a project does not use CMake as its build system?**

Installation Process Issues

- **Cumbersome, space and time wasting**
- Most packages are **not ready to install several versions** of the same product at once
  - on install they do not produce *subdirectories for each package version*
  - big pain if *your projects require specific but different versions* of dependencies
- **What if a project does not use CMake as its build system?**

Just Use Package Managers!

C++Now 2021 Keynote

CONAN

THE PATH TOWARDS 2.0

Feedback from my customer

Feedback from my customer

Conan just works :-)

-- Happy Customer

Feedback from my customer

Conan just works :-)

-- Happy Customer

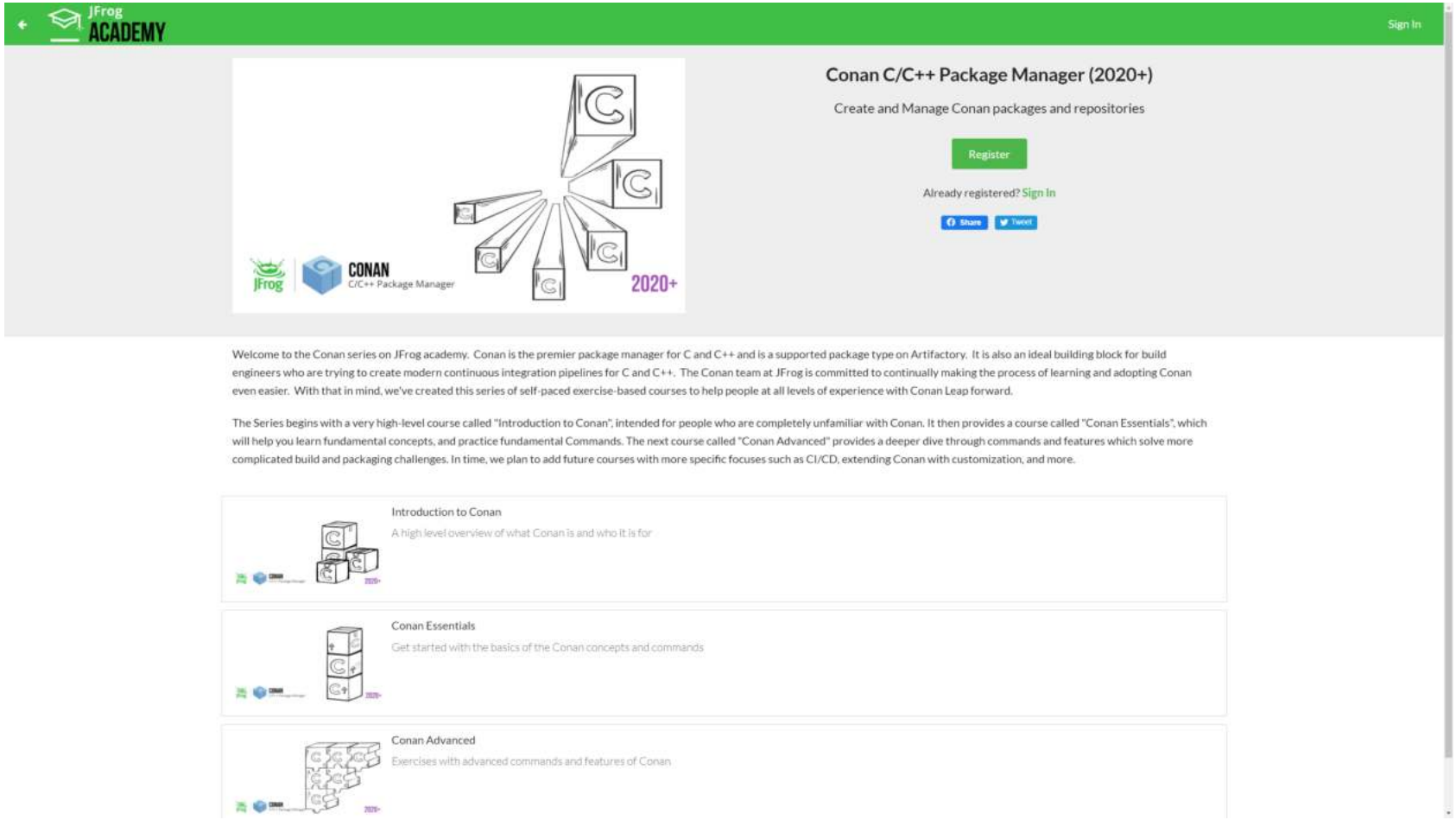
How often do you hear something like that about the C++ ecosystem?

Conan Documentation

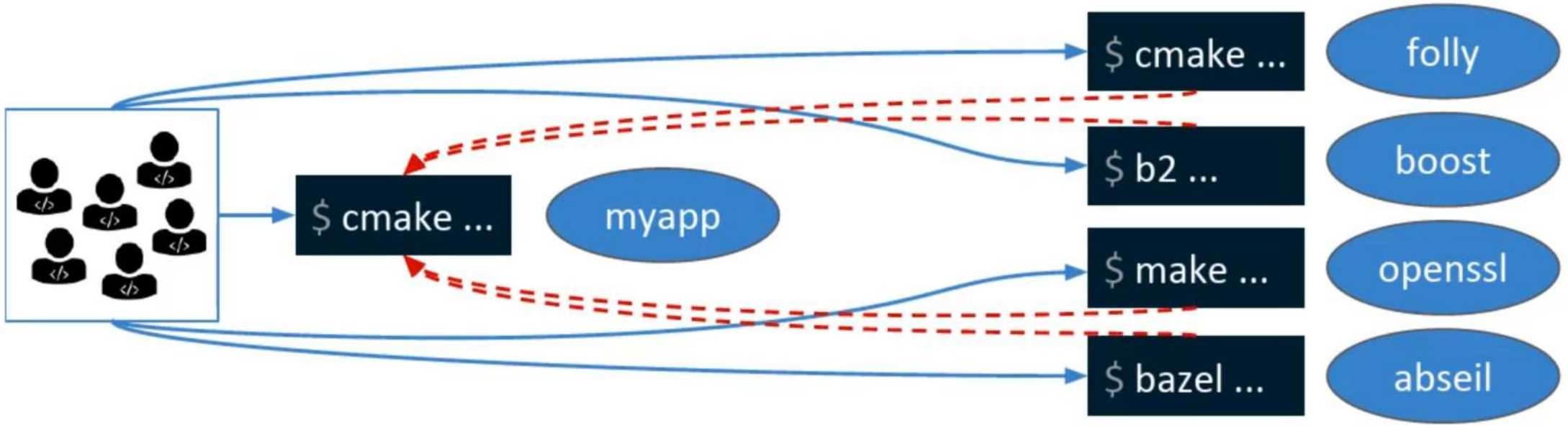
Conan client's interface is really simple and intuitive to use.

- In case on any doubts refer to
  - **`conan [command] -h`**
  - https://docs.conan.io
    - a high-quality project documentation
    - no additional book needed

Free Training Courses

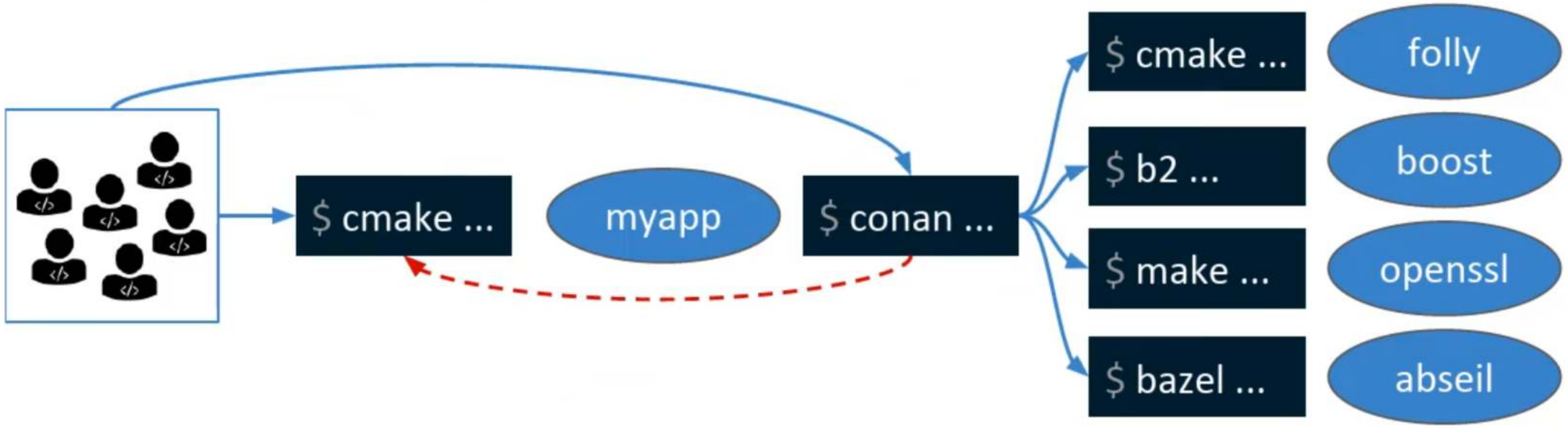


Abstracting Away Build Systems



Besides using the preferred build system (i.e. CMake) **users need to learn and interact** with each build system of every dependency.

Abstracting Away Build Systems



Users interact only with the package manager and if needed with the preferred build system (i.e. CMake).

How to enable my project with Conan?

How to enable my project with Conan?

1. Install Conan

How to enable my project with Conan?

1. Install Conan
2. Set a development profile

How to enable my project with Conan?

1. Install Conan
2. Set a development profile
3. *[Optional]* Add custom remotes

How to enable my project with Conan?

1. Install Conan
2. Set a development profile
3. *[Optional]* Add custom remotes
4. Create a **`conanfile`**

How to enable my project with Conan?

1. Install Conan
2. Set a development profile
3. *[Optional]* Add custom remotes
4. Create a **`conanfile`**
5. Provide dependencies with Conan

Conan Installation/Upgrade

```
pip3 install -U conan
```

Conan Profile

- During the first execution Conan *initializes its configuration*
  - **~/.conan** directory by default

Conan Profile

- During the first execution Conan *initializes its configuration*
  - **~/.conan** directory by default
- Among others a *default profile is created* based on the *auto-detected* toolchain
  - **~/.conan/profiles/default**
  - just a text file

Conan Profile

- During the first execution Conan *initializes its configuration*
  - **~/.conan** directory by default
- Among others a *default profile is created* based on the *auto-detected* toolchain
  - **~/.conan/profiles/default**
  - just a text file

```
conan profile show default
```

A `default` profile

`~/.conan/profiles/default`

```
[settings]
os=Linux
os_build=Linux
arch=x86_64
arch_build=x86_64
compiler=gcc
compiler.version=10
compiler.libcxx=libstdc++
build_type=Release
[options]
[build_requires]
[env]
```

A `default` profile

`~/.conan/profiles/default`

```
[settings]
os=Linux
os_build=Linux
arch=x86_64
arch_build=x86_64
compiler=gcc
compiler.version=10
compiler.libcxx=libstdc++
build_type=Release
[options]
[build_requires]
[env]
```

Because of backward compatibility reasons, until Conan 2.0, **`libstdc++`** is selected by default.

Amend the library version

- Either *manually edit* the profile file or *use Conan client* to do the work

GCC

```
conan profile update settings.compiler.libcxx=libstdc++11 default
```

Amend the library version

- Either *manually edit* the profile file or *use Conan client* to do the work

GCC

```
conan profile update settings.compiler.libcxx=libstdc++11 default
```

Conan 2.0 will be selecting the **`libstdc++11`** by default for gcc >= 5.0.

Amend the library version

- Either *manually edit* the profile file or *use Conan client* to do the work

GCC

```
conan profile update settings.compiler.libcxx=libstdc++11 default
```

CLANG

```
conan profile update settings.compiler.libcxx=libc++ default
```

Conan 2.0 will be selecting the **`libstdc++11`** by default for gcc >= 5.0.

Forcing a specific C++ version

```
conan profile update settings.compiler.cppstd=20 default
```

```
[settings]
os=Linux
os_build=Linux
arch=x86_64
arch_build=x86_64
compiler=gcc
compiler.version=10
compiler.cppstd=20
compiler.libcxx=libstdc++11
build_type=Release
[options]
[build_requires]
[env]
```

Forcing a specific C++ version

```
conan profile update settings.compiler.cppstd=20 default
```

```
[settings]
os=Linux
os_build=Linux
arch=x86_64
arch_build=x86_64
compiler=gcc
compiler.version=10
compiler.cppstd=20
compiler.libcxx=libstdc++11
build_type=Release
[options]
[build_requires]
[env]
```

- Alternatively

```
conan install .. -s compiler.cppstd=20
```

Creating a set of development profiles

- Copy-paste a default profile and adjust it to your preferences

```
conan profile list
```

```
clang11
clang12
clang12-libstdc++11
clang13
default
gcc10
gcc7
gcc8
gcc9
```

Creating a set of development profiles

- Copy-paste a default profile and adjust it to your preferences

```
conan profile list
```

```
clang11
clang12
clang12-libstdc++11
clang13
default
gcc10
gcc7
gcc8
gcc9
```

```
ls -1 ~/.conan/profiles/
```

```
clang11
clang12
clang12-libstdc++11
clang13
default
gcc10
gcc7
gcc8
gcc9
```

Creating a set of development profiles

- Copy-paste a default profile and adjust it to your preferences

```
conan profile list
```

```
clang11
clang12
clang12-libstdc++11
clang13
default
gcc10
gcc7
gcc8
gcc9
```

```
ls -1 ~/.conan/profiles/
```

```
clang11
clang12
clang12-libstdc++11
clang13
default
gcc10
gcc7
gcc8
gcc9
```

- Debug versions can be easily derived with

```
conan install .. -pr <profile_name> -s build_type=Debug
```

Conan profile example

GCC-10

```
[settings]
os=Linux
os_build=Linux
arch=x86_64
arch_build=x86_64
compiler=gcc
compiler.version=10
compiler.cppstd=20
compiler.libcxx=libstdc++11
build_type=Release

[env]
CXX=/usr/bin/g++-10
CC=/usr/bin/gcc-10
```

Conan profile example

CLANG-12

```
[settings]
os=Linux
os_build=Linux
arch=x86_64
arch_build=x86_64
compiler=clang
compiler.version=12
compiler.cppstd=20
compiler.libcxx=libc++
build_type=Release

[env]
CXX=/usr/bin/clang++-12
CC=/usr/bin/clang-12
```

Conan profile example

VISUAL STUDIO

```
[settings]
os=Windows
os_build=Windows
arch=x86_64
arch_build=x86_64
compiler=Visual Studio
compiler.version=16
compiler.cppstd=20
build_type=Release
```

Conan profile example

MINGW

```
[build_requires]
msys2/20200517

[settings]
os_build=Windows
os=Windows
arch=x86_64
arch_build=x86_64
compiler=gcc
compiler.version=9.3
compiler.exception=seh
compiler.libcxx=libstdc++11
compiler.threads=posix
build_type=Release
```

Conan profile example

cross_compile

```
CROSS_GCC=arm-linux-gnueabihf

include(default)

[settings]
arch=armv7

[env]
CC=$CROSS_GCC-gcc
CXX=$CROSS_GCC-g++
```

Conan profile example

cross_compile

```
CROSS_GCC=arm-linux-gnueabihf

include(default)

[settings]
arch=armv7

[env]
CC=$CROSS_GCC-gcc
CXX=$CROSS_GCC-g++
```

- Conan profile files *can be distributed with the project source code*

```
conan install .. -pr=../cross_compile
```

Overwrite setting or env for specific package

```
[settings]
os=Linux
compiler=gcc
compiler.version=4.9
compiler.libcxx=libstdc++
build_type=Release
arch=armv7
OpenSSL:compiler.version=4.8

[env]
CC=arm-linux-gnueabihf-gcc
CXX=arm-linux-gnueabihf-g++
zlib:CC=arm-linux-gnuabihf-gcc-patched
```

Profiles composition

```
conan install . -pr=windows -pr=vs2017
conan install . -pr=windows -pr=vs2017 -s build_type=Debug
conan create . -pr=windows -pr=vs2017
```

- Multiple profiles can be passed at CLI
- Conan reconciles all the values from all the profiles

Sharing Conan configuration

```
conan config install http://github.com/user/conan_config/.git
```

- In a company or organization it might be important to have a **common Conan configuration**

Sharing Conan configuration

```
conan config install http://github.com/user/conan_config/.git
```

- In a company or organization it might be important to have a common Conan configuration
- Obtains the configuration and installs in the local Conan file system
- Either one specific file or a set of configuration files

Sharing Conan configuration

```
conan config install http://github.com/user/conan_config/.git
```

- In a company or organization it might be important to have a common Conan configuration
- Obtains the configuration and installs in the local Conan file system
- Either one specific file or a set of configuration files
- Can download from
  - a local or remote zip file
  - a local directory
  - a git repository

Sharing Conan configuration

```
conan config install http://github.com/user/conan_config/.git
```

- In a company or organization it might be important to have a **common Conan configuration**
- **Obtains** the configuration **and installs** in the local Conan file system
- Either **one specific file or a set of configuration files**
- **Can download from**
  - a local or remote zip file
  - a local directory
  - a git repository

Only the files that are shared will be replaced.

ConanCenter

- **Public** Conan repository for **OSS packages**

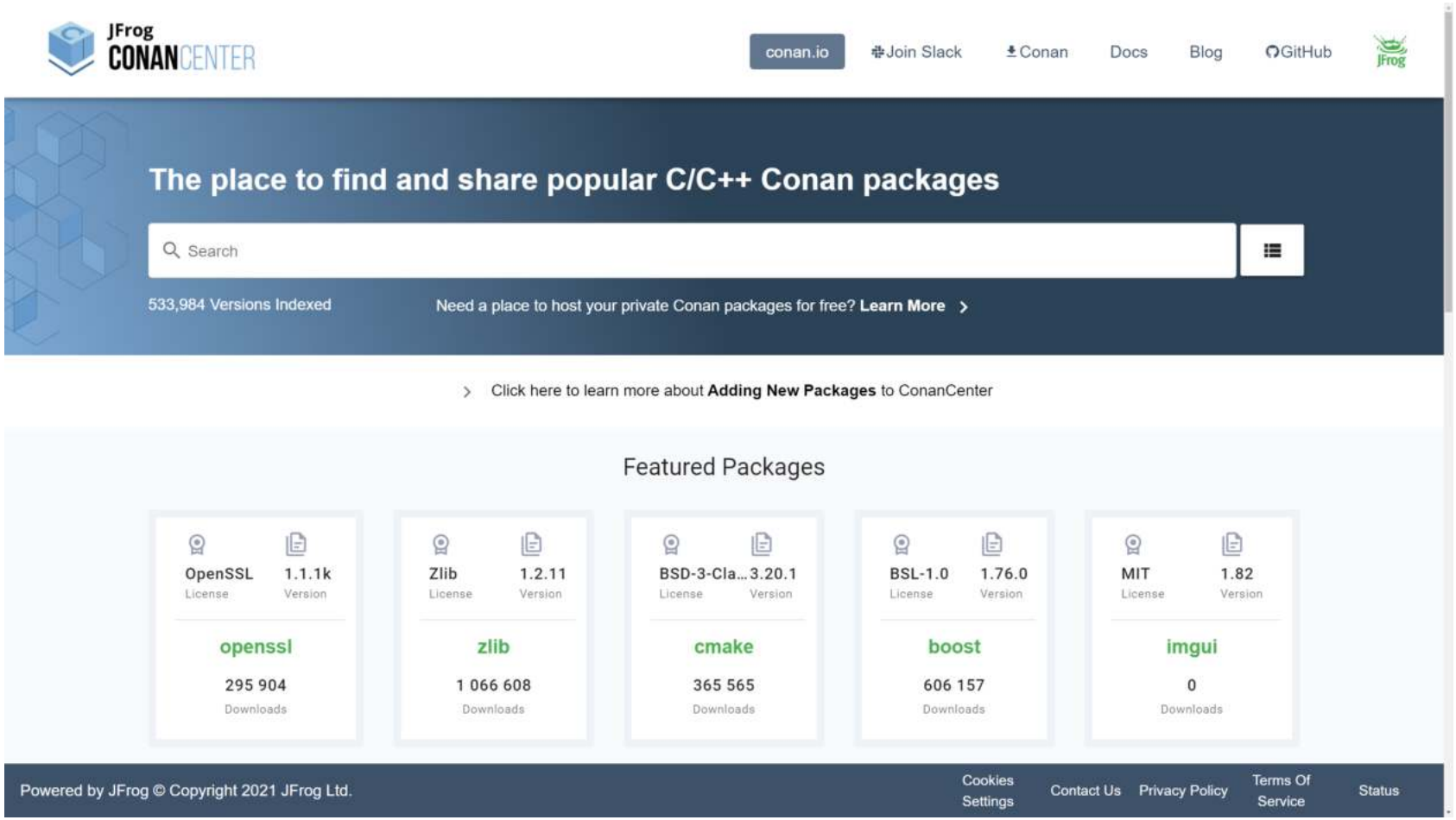
ConanCenter

- **Public** Conan repository for **OSS packages**
- **Moderated** and **maintained** by the Conan team
- Contributions by the **community**

ConanCenter

- **Public** Conan repository for **OSS packages**
- **Moderated** and **maintained** by the Conan team
- Contributions by the **community**
- Contains **hundreds** of popular libraries and applications
- The recipes for these packages can be found at **ConanCenterIndex**
  - https://github.com/conan-io/conan-center-index
  - integrated CI automatically builds Conan packages for each PR

ConanCenter



Conan Custom Remotes

- Conan is a **decentralized** package manager
- **Custom remotes** allow to host
  - *Live At Head* public packages
  - packages *not meant for a larger audience*
  - prebuilt artifacts for *non-standard configurations*
  - *private* packages

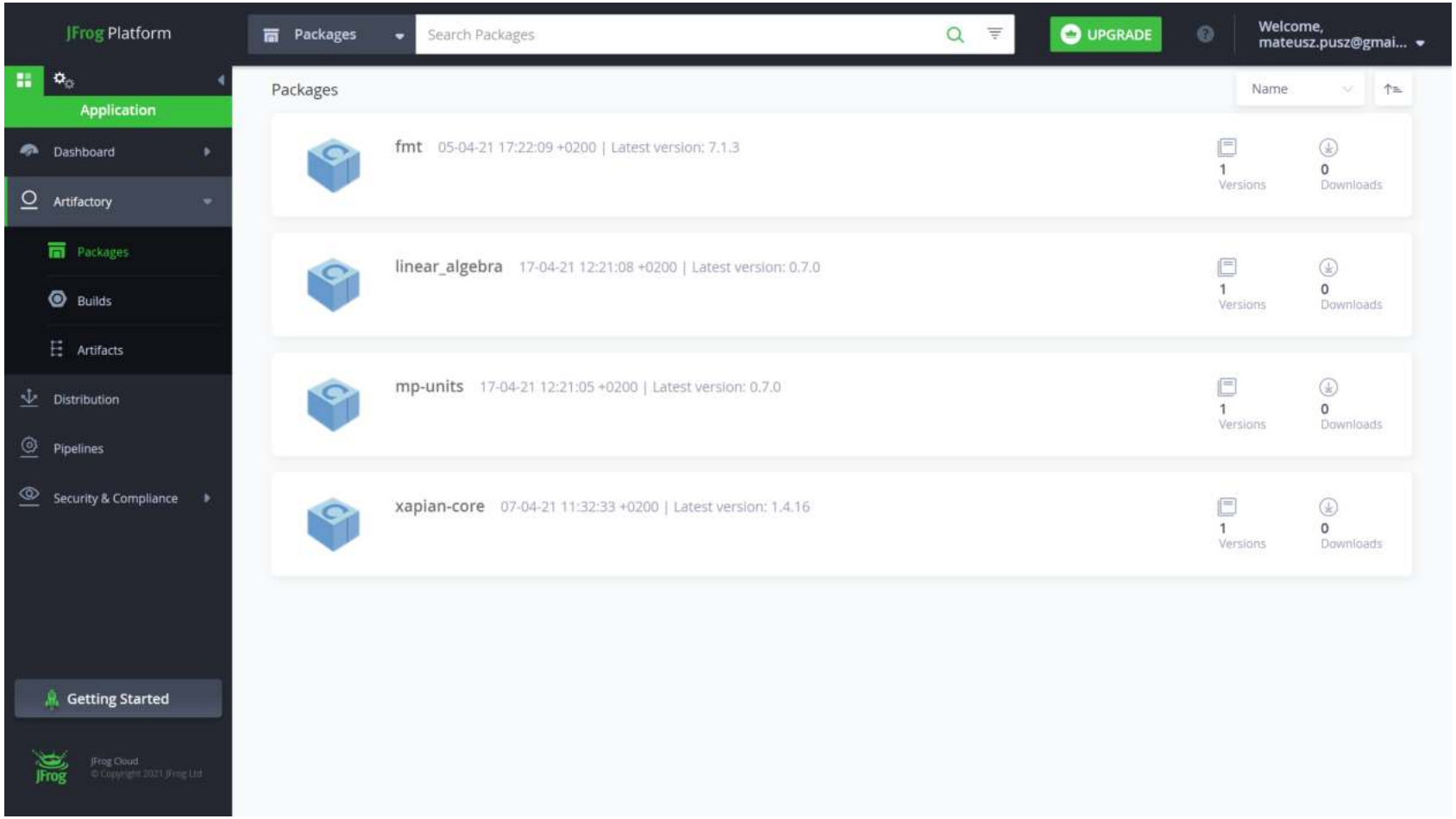
Conan Custom Remotes

- Conan is a **decentralized** package manager
- **Custom remotes** allow to host
  - *Live At Head* public packages
  - packages *not meant for a larger audience*
  - prebuilt artifacts for *non-standard configurations*
  - *private* packages

JFROG ARTIFACTORY

- Self-managed
- **Cloud** or **On-Premise**
- **JFrog Artifactory Community Edition for C++** makes it easy to start with own server for free

JFrog Cloud


JFrog Platform
Packages
Search Packages
UPGRADE
Welcome, mateusz.pusz@gmai...
Application
Dashboard
Artifactory
Packages
Builds
Artifacts
Distribution
Pipelines
Security & Compliance
Getting Started
JFrog Cloud
Packages
Name
fmt 05-04-21 17:22:09 +0200 | Latest version: 7.1.3
1 Versions
0 Downloads
linear_algebra 17-04-21 12:21:08 +0200 | Latest version: 0.7.0
1 Versions
0 Downloads
mp-units 17-04-21 12:21:05 +0200 | Latest version: 0.7.0
1 Versions
0 Downloads
xapian-core 07-04-21 11:32:33 +0200 | Latest version: 1.4.16
1 Versions
0 Downloads

Conan Remotes

```
conan remote list
```

```
conan-mpusz: https://mpusz.jfrog.io/artifactory/api/conan/conan-oss [Verify SSL: True]
conan-train-it: https://trainit.jfrog.io/artifactory/api/conan/conan-trainings [Verify SSL: True]
conan-center: https://center.conan.io [Verify SSL: True]
linear-algebra: https://twonington.jfrog.io/artifactory/api/conan/conan-oss [Verify SSL: True]
```

Conan Remotes

```
conan remote list
```

```
conan-mpusz: https://mpusz.jfrog.io/artifactory/api/conan/conan-oss [Verify SSL: True]
conan-train-it: https://trainit.jfrog.io/artifactory/api/conan/conan-trainings [Verify SSL: True]
conan-center: https://center.conan.io [Verify SSL: True]
linear-algebra: https://twonington.jfrog.io/artifactory/api/conan/conan-oss [Verify SSL: True]
```

The above URL for ConanCenter has BETA status as of now but most probably will replace currently used https://conan.bintray.com (Bintray is EOL).

Conan File

conanfile.txt

```
[requires]
mp-units/0.7.0@mpusz/testing
boost/1.75.0

[options]
boost:header_only=True

[generators]
CMakeToolchain
CMakeDeps
```

Conan File

conanfile.txt

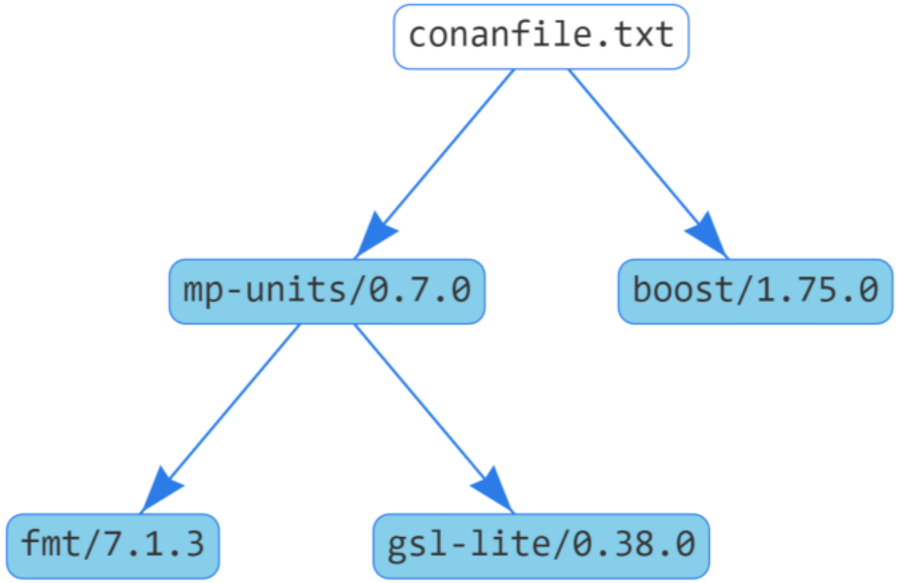
```
[requires]
mp-units/0.7.0@mpusz/testing
boost/1.75.0

[options]
boost:header_only=True

[generators]
CMakeToolchain
CMakeDeps
```

CMakeToolchain and **CMakeDeps** are experimental CMake generators targeting Conan 2.0.

Conan File

conanfile.txt

```
[requires]
mp-units/0.7.0@mpusz/testing
boost/1.75.0

[options]
boost:header_only=True

[generators]
CMakeToolchain
CMakeDeps
```



CMakeToolchain and **CMakeDeps** are experimental CMake generators targeting Conan 2.0.

Conan Recipe Reference

Recipe reference has a form of a string and is an identifier of the package in Conan.

Conan Recipe Reference

Recipe reference has a form of a string and is an identifier of the package in Conan.

- One recipe reference can represent **any number of separate physical binaries**
  - each binary being the result of each unique build of the library or application in the package

Conan Recipe Reference

Recipe reference has a form of a string and is an identifier of the package in Conan.

- One recipe reference can represent any number of separate physical binaries
  - each binary being the result of each unique build of the library or application in the package

A single identifier being able to represent any number of unique binaries is one of Conan major innovations.

Conan Recipe Reference (Short)

```
package_name/package_version
```

PACKAGE_NAME

- Usually project/library name

PACKAGE_VERSION

- Any string

Conan Recipe Reference (Short)

```
package_name/package_version
```

PACKAGE_NAME

- Usually project/library name

PACKAGE_VERSION

- Any string

Short version is typically reserved for packages from Conan Center.

Conan Recipe Reference (Short)

```
package_name/package_version
```

PACKAGE_NAME

- Usually project/library name

PACKAGE_VERSION

- Any string

Often spelled in the command line with trailing `@` (i.e. **`gtest/1.10.0@`**).
Needed to disambiguate from a regular system path.

Conan Recipe Reference (Normal)

```
package_name/package_version@owner/channel
```

PACKAGE_NAME

- Usually project/library name

PACKAGE_VERSION

- Any string

OWNER

- Owner of the package version
- Namespace that allows different users to have their *own packages for the same library* with the same name

CHANNEL

- Allows *different packages* for the same library
- Usually denote the maturity of a package ("stable", "testing")

Conan Recipe Reference (Full)

```
package_name/package_version@owner/channel#RREV:PACKAGE_ID#PREV
```

PACKAGE_ID

- Package identifier
- The hash created from the ***`settings`***, ***`options`***, *and* ***`requires`***

RREV

- Recipe revision
- The hash of *contents of the recipe*

PREV

- Package revision
- The hash of the *contents of the package*

Installing dependencies with Conan

```
mkdir build && cd build
conan install .. -pr <your_conan_profile>
conan install .. -pr <your_conan_profile> -s build_type=Debug
cmake .. -DCMAKE_TOOLCHAIN_FILE=conan_toolchain.cmake
cmake --build .
cmake --build . --config Debug
```

Ninja Multi-Config and CMakeDeps work great together!

Installing dependencies with Conan

```
mkdir build && cd build
conan install .. -pr <your_conan_profile>
conan install .. -pr <your_conan_profile> -s build_type=Debug
cmake .. -DCMAKE_TOOLCHAIN_FILE=conan_toolchain.cmake
cmake --build .
cmake --build . --config Debug
```

Installing dependencies with Conan

```
mkdir build && cd build
conan install .. -pr <your_conan_profile>
conan install .. -pr <your_conan_profile> -s build_type=Debug
cmake .. -DCMAKE_TOOLCHAIN_FILE=conan_toolchain.cmake
cmake --build .
cmake --build . --config Debug
```

- **`-pr <your_conan_profile>`** option *can be skipped if a* ***`default`*** profile should be used

Installing dependencies with Conan

```
mkdir build && cd build
conan install .. -pr <your_conan_profile>
conan install .. -pr <your_conan_profile> -s build_type=Debug
cmake .. -DCMAKE_TOOLCHAIN_FILE=conan_toolchain.cmake
cmake --build .
cmake --build . --config Debug
```

- **`-pr <your_conan_profile>`** option *can be skipped if a **`default`*** profile should be used
- ***`CMAKE_INSTALL_PREFIX`*** *not needed anymore* during CMake configuration step
  - Conan automatically provides a path for each dependency

Installing dependencies with Conan

```
mkdir build && cd build
conan install .. -pr <your_conan_profile>
conan install .. -pr <your_conan_profile> -s build_type=Debug
cmake .. -DCMAKE_TOOLCHAIN_FILE=conan_toolchain.cmake
cmake --build .
cmake --build . --config Debug
```

- **`-pr <your_conan_profile>`** option *can be skipped if a **`default`*** profile should be used
- ***`CMAKE_INSTALL_PREFIX`** not needed anymore* during CMake configuration step
  - Conan automatically provides a path for each dependency

Besides providing a CMake toolchain file generated by Conan, **no additional Conan-specific changes are needed** to CMake workflow or its files.

Conan Install

By default Conan will always look for a prebuilt binary to use and will fail if it is not found.

Conan Install

By default Conan will always look for a prebuilt binary to use and will fail if it is not found.

- Conan **does not** silently **build from sources**
- Helps to maintain a fast CI pipeline
  - **build once use forever**
  - exactly **the same binary artifacts** always used for all dependents

Conan Install

By default Conan will always look for a prebuilt binary to use and will fail if it is not found.

- Conan **does not** silently **build from sources**
- Helps to maintain a fast CI pipeline
  - **build once use forever**
  - exactly **the same binary artifacts** always used for all dependents

Build the dependency from its sources once and upload the resulting binary package to the Conan Artifactory for future CI builds.

Conan Building Options

```
conan install -b XXX ...
```

Conan Building Options

```
conan install -b XXX ...
```

<none>

- Forces *(re)building all packages from sources*
  - even if prebuilt binaries exist in local cache

Conan Building Options

```
conan install -b XXX ...
```

<none>

- Forces *(re)building all packages from sources*
  - even if prebuilt binaries exist in local cache

never (DEFAULT)

- *Never build* from sources
  - fails when a binary package is not available

Conan Building Options

```
conan install -b XXX ...
```

<none>

- Forces *(re)building all packages from sources*
  - even if prebuilt binaries exist in local cache

never (DEFAULT)

- *Never build* from sources
  - fails when a binary package is not available

missing

- Build only those packages that *do not have prebuilt binaries*

Conan Building Options

```
conan install -b XXX ...
```

<none>

- Forces *(re)building all packages from sources*
  - even if prebuilt binaries exist in local cache

never (DEFAULT)

- *Never build* from sources
  - fails when a binary package is not available

missing

- Build only those packages that *do not have prebuilt binaries*

outdated

- Only build packages *without binaries or if they were built for an older version* of the recipe

Conan Building Options

```
conan install -b XXX ...
```

<none>

- Forces *(re)building all packages from sources*
  - even if prebuilt binaries exist in local cache

never (DEFAULT)

- *Never build* from sources
  - fails when a binary package is not available

missing

- Build only those packages that *do not have prebuilt binaries*

outdated

- Only build packages *without binaries or if they were built for an older version* of the recipe

cascade

- Build packages that have *at least one dependency being built from source*

Conan Building Options

```
conan install -b XXX ...
```

<none>

- Forces *(re)building all packages from sources*
  - even if prebuilt binaries exist in local cache

never (DEFAULT)

- *Never build* from sources
  - fails when a binary package is not available

missing

- Build only those packages that *do not have prebuilt binaries*

outdated

- Only build packages *without binaries or if they were built for an older version* of the recipe

cascade

- Build packages that have *at least one dependency being built from source*

<pattern>

- Every matching package *will be (re)built*
- **!<pattern>** *excludes* from (re)build

Conan Local Cache

All package related data lands in the Conan Local Cache.

Conan Local Cache

All package related data lands in the Conan Local Cache.

- Typically **located in the `~/.conan/data`** directory
  - default path can be overridden with the **`CONAN_USER_HOME`** environment variable

Conan Local Cache

All package related data lands in the Conan Local Cache.

- Typically **located in the `~/.conan/data`** directory
  - default path can be overridden with the **`CONAN_USER_HOME`** environment variable
- **Conan is the one responsible for housekeeping of different ABI versions** of the same package
  - *rather than handled manually* by the user via a CMake install step and **`CMAKE_INSTALL_PREFIX`**

Conan Local Cache

All package related data lands in the Conan Local Cache.

- Typically **located in the `~/.conan/data`** directory
  - default path can be overridden with the **`CONAN_USER_HOME`** environment variable
- **Conan is the one responsible for housekeeping of different ABI versions** of the same package
  - *rather than handled manually* by the user via a CMake install step and **`CMAKE_INSTALL_PREFIX`**

Conan Local Cache helps in sharing the same dependencies between various projects and repositories on the same PC.

Conan Generators

Conan Generators

`CMakeToolchain`

- Toolchain generator for CMake
- Helps to achieve **exactly the same build** via local development flows and when the package is created in the cache by Conan

Conan Generators

`CMakeToolchain`

- Toolchain generator for CMake
- Helps to achieve **exactly the same build** via local development flows and when the package is created in the cache by Conan
- Generates
  - `conan_toolchain.cmake` - *translates* the current package configuration, settings, and options into CMake toolchain syntax

Conan Generators

`CMakeToolchain`

- Toolchain generator for CMake
- Helps to achieve **exactly the same build** via local development flows and when the package is created in the cache by Conan
- Generates
  - `conan_toolchain.cmake` - *translates* the current package configuration, settings, and options into CMake toolchain syntax
  - `conanbuild.json` - contains *arguments* to the command line **`CMake()`** helper used in the recipe **`build()`** method (i.e. CMake generator)

Conan Generators

`CMakeToolchain`

- Toolchain generator for CMake
- Helps to achieve **exactly the same build** via local development flows and when the package is created in the cache by Conan
- Generates
  - `conan_toolchain.cmake` - *translates* the current package configuration, settings, and options into CMake toolchain syntax
  - `conanbuild.json` - contains *arguments* to the command line **`CMake()`** helper used in the recipe **`build()`** method (i.e. CMake generator)
  - `conanvcvars.bat` - for *MSVC compilers* and CMake generators like Ninja or NMake

Conan Generators

`CMakeDeps`

- Multi-configuration generator

Conan Generators

`CMakeDeps`

- Multi-configuration generator
- **Generates one `xxxx-config.cmake` file per dependency** together with other necessary **`.cmake`** files like version, flags, and directory data or configuration

Conan Generators

`CMakeDeps`

- Multi-configuration generator
- **Generates one `xxxx-config.cmake` file per dependency** together with other necessary **`.cmake`** files like version, flags, and directory data or configuration
- Enables CMake's **`CONFIG` mode** for packages that
  - do not use CMake at all
  - do not provide installation support
  - generate broken CMake configuration scripts

Conan Generators

CMakeDeps

- Multi-configuration generator
- **Generates one `xxxx-config.cmake` file per dependency** together with other necessary **`.cmake`** files like version, flags, and directory data or configuration
- Enables CMake's **`CONFIG` mode** for packages that
  - do not use CMake at all
  - do not provide installation support
  - generate broken CMake configuration scripts

It is encouraged to depend on Conan generated files rather than trying to use packages' original files (if provided).

Conan Generators

`deploy`

- **Copies folders** of the package and all the dependencies in a graph
- Can be used to **deploy binaries** from the local cache to the user space
- Each package will land in a directory named with the package name
- Generates **`deploy_manifest.txt`** file
  - contains the *list of all the files deployed and hashes* of their contents

Conan Generators

`deploy`

- **Copies folders** of the package and all the dependencies in a graph
- Can be used to **deploy binaries** from the local cache to the user space
- Each package will land in a directory named with the package name
- Generates **`deploy_manifest.txt`** file
  - contains the *list of all the files deployed and hashes* of their contents

`--install-folder` parameter can be used to output the contents of the packages to a specific folder.

Conan Generators

`json`

- Generates **`conanbuildinfo.json`** file
- *Contains all information about the binary package and its dependencies*
  - include, lib, binary directories
  - preprocessor defines
  - compilation and linking flags
  - libraries
  - ...
- It is mostly useful **for custom integrations**

Conan recipe on steroids

conanfile.txt

```
[requires]
mp-units/0.7.0@mpusz/testing
boost/1.75.0

[options]
boost:header_only=True

[generators]
CMakeToolchain
CMakeDeps
```

conanfile.py

```
from conans import ConanFile

class MyProjectConan(ConanFile):
    name = "my_project"
    settings = "os", "compiler", "build_type", "arch"
    requires = (
        "mp-units/0.7.0@mpusz/testing",
        "boost/1.75.0"
    )
    default_options = { "boost:header_only=True" }
    generators = "CMakeToolchain", "CMakeDeps"
```

More power with a conanfile.py

```
from conans import ConanFile
from conan.tools.cmake import CMake

class MyProjectConan(ConanFile):
    name = "my_project"
    settings = "os", "compiler", "build_type", "arch"
    requires = (
        "mp-units/0.7.0@mpusz/testing",
        "boost/1.75.0"
    )
    default_options = { "boost:header_only=True" }
    generators = "CMakeToolchain", "CMakeDeps"

    def build(self):
        cmake = CMake(self)
        cmake.configure()
        cmake.build()
```

More power with a conanfile.py

```
from conans import ConanFile
from conan.tools.cmake import CMake

class MyProjectConan(ConanFile):
    name = "my_project"
    settings = "os", "compiler", "build_type", "arch"
    requires = (
        "mp-units/0.7.0@mpusz/testing",
        "boost/1.75.0"
    )
    default_options = { "boost:header_only=True" }
    generators = "CMakeToolchain", "CMakeDeps"

    def build(self):
        cmake = CMake(self)
        cmake.configure()
        cmake.build()
```

```
conan install .. -pr <your_conan_profile>
conan build ..
```

Starting a new library project

```
conan new -m v2_cmake my_project/0.1.0
```

```
File saved: conanfile.py
File saved: src/CMakeLists.txt
File saved: src/my_project.cpp
File saved: src/my_project.h
File saved: test_package/CMakeLists.txt
File saved: test_package/conanfile.py
File saved: test_package/example.cpp
```

Starting a new library project

conanfile.py

```
from conans import ConanFile
from conan.tools.cmake import CMakeToolchain, CMake

class MyProjectConan(ConanFile):
    name = "my_project"
    version = "0.1.0"
    license = "<Put the package license here>"
    author = "<Put your name here> <And your email here>"
    url = "<Package recipe repository url here>"
    description = "<Description of MyProject here>"
    topics = ("<Put some tag here>", "<and here>")
    settings = "os", "compiler", "build_type", "arch"
    options = {
      "shared": [True, False],
      "fPIC": [True, False]
    }
    default_options = {"shared": False, "fPIC": True}
    exports_sources = "src/*"
```

Starting a new library project

conanfile.py

```
from conans import ConanFile
from conan.tools.cmake import CMakeToolchain, CMake

class MyProjectConan(ConanFile):
    name = "my_project"
    version = "0.1.0"
    license = "<Put the package license here>"
    author = "<Put your name here> <And your email here>"
    url = "<Package recipe repository url here>"
    description = "<Description of MyProject here>"
    topics = ("<Put some tag here>", "<and here>")
    settings = "os", "compiler", "build_type", "arch"
    options = {
      "shared": [True, False],
      "fPIC": [True, False]
    }
    default_options = {"shared": False, "fPIC": True}
    exports_sources = "src/*"
```

```
    def config_options(self):
        if self.settings.os == "Windows":
            del self.options.fPIC

    def generate(self):
        tc = CMakeToolchain(self)
        tc.generate()

    def build(self):
        cmake = CMake(self)
        cmake.configure(source_folder="src")
        cmake.build()

    def package(self):
        self.copy("*.h", dst="include", src="src")
        self.copy("*.lib", dst="lib", keep_path=False)
        self.copy("*.dll", dst="bin", keep_path=False)
        self.copy("*.dylib*", dst="lib", keep_path=False)
        self.copy("*.so", dst="lib", keep_path=False)
        self.copy("*.a", dst="lib", keep_path=False)

    def package_info(self):
        self.cpp_info.libs = ["my_project"]
```

Starting a new library project

test_package/conanfile.py

```
import os

from conans import ConanFile, tools
from conan.tools.cmake import CMake

class MyProjectTestConan(ConanFile):
    settings = "os", "compiler", "build_type", "arch"
    generators = "CMakeDeps", "CMakeToolchain", "VirtualEnv"
    apply_env = False

    def build(self):
        cmake = CMake(self)
        cmake.configure()
        cmake.build()

    def test(self):
        if not tools.cross_building(self):
            self.run(os.path.sep.join([".", "bin", "example"]), env="conanrunenv")
```

Defining package ABI compatibility

- By default **Conan creates a unique package identifier** for every unique configuration
  - settings, options, requires

Defining package ABI compatibility

- By default **Conan creates a unique package identifier** for every unique configuration
  - settings, options, requires
- Each unique package identifier is assumed to **specify a different ABI** of the same package

Defining package ABI compatibility

- By default **Conan creates a unique package identifier** for every unique configuration
  - settings, options, requires
- Each unique package identifier is assumed to **specify a different ABI** of the same package
- If a **package creation** for a specific configuration **results with a new identifier** then its **artifacts are stored in a separate directory** in the Conan artifacts database
  - Conan Local Cache
  - Artifactory

Defining a custom `package_id()`

- **`package_id()`** method can be overridden to control the package ID generation

```
def package_id(self):
    del self.info.settings.compiler.cppstd
    del self.info.options.build_docs
```

Defining a custom package_id()

- **package_id()** method can be overridden to control the package ID generation

```
def package_id(self):
    del self.info.settings.compiler.cppstd
    del self.info.options.build_docs
```

- Special way to define a header-only library

```
def package_id(self):
    self.info.header_only()
```

Defining a custom package_id()

- **package_id()** method can be overridden to control the package ID generation

```
def package_id(self):
    del self.info.settings.compiler.cppstd
    del self.info.options.build_docs
```

- Special way to define a header-only library

```
def package_id(self):
    self.info.header_only()
```

User does not have to guess how many variants of a package to keep in the installation paths anymore.

Using Components

- Often a package **contains more than one library**

Using Components

- Often a package **contains more than one library**
- Usually **libraries** inside the same package **depend on each other**
  - modelling the relationship among them is required

Using Components

- Often a package contains more than one library
- Usually libraries inside the same package depend on each other
  - modelling the relationship among them is required
- **`package_info()`** method can be overridden to provide such information to CMakeDeps

```
def package_info(self):
    self.cpp_info.name = "OpenSSL"
    self.cpp_info.components["crypto"].libs = ["libcrypto"]
    self.cpp_info.components["crypto"].defines = ["DEFINE_CRYPTO=1"]
    self.cpp_info.components["ssl"].includedirs = ["include/headers_ssl"]
    self.cpp_info.components["ssl"].libs = ["libssl"]
    self.cpp_info.components["ssl"].requires = ["crypto"]
```

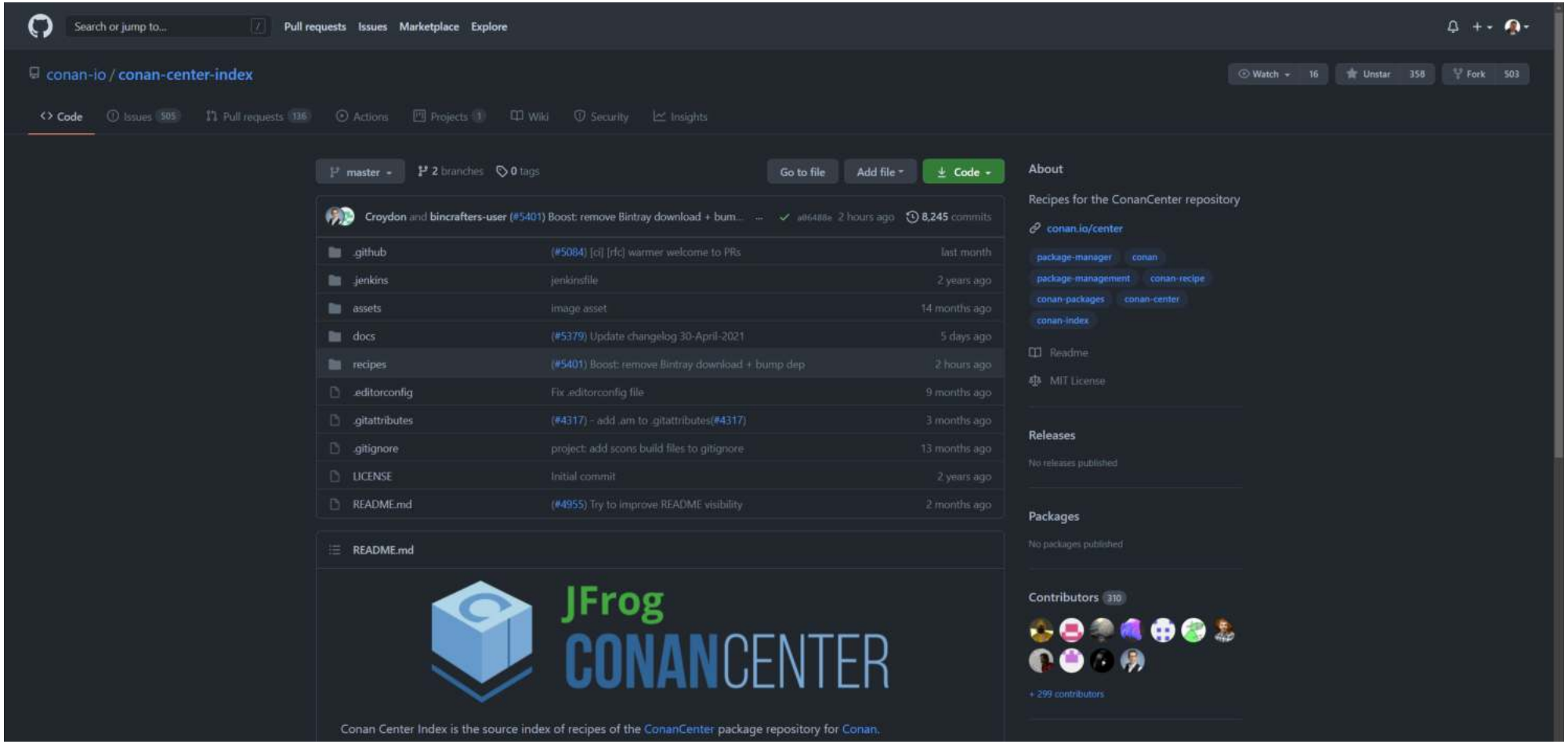
Using Components

- Often a package contains more than one library
- Usually libraries inside the same package depend on each other
  - modelling the relationship among them is required
- **`package_info()`** method can be overridden to provide such information to CMakeDeps

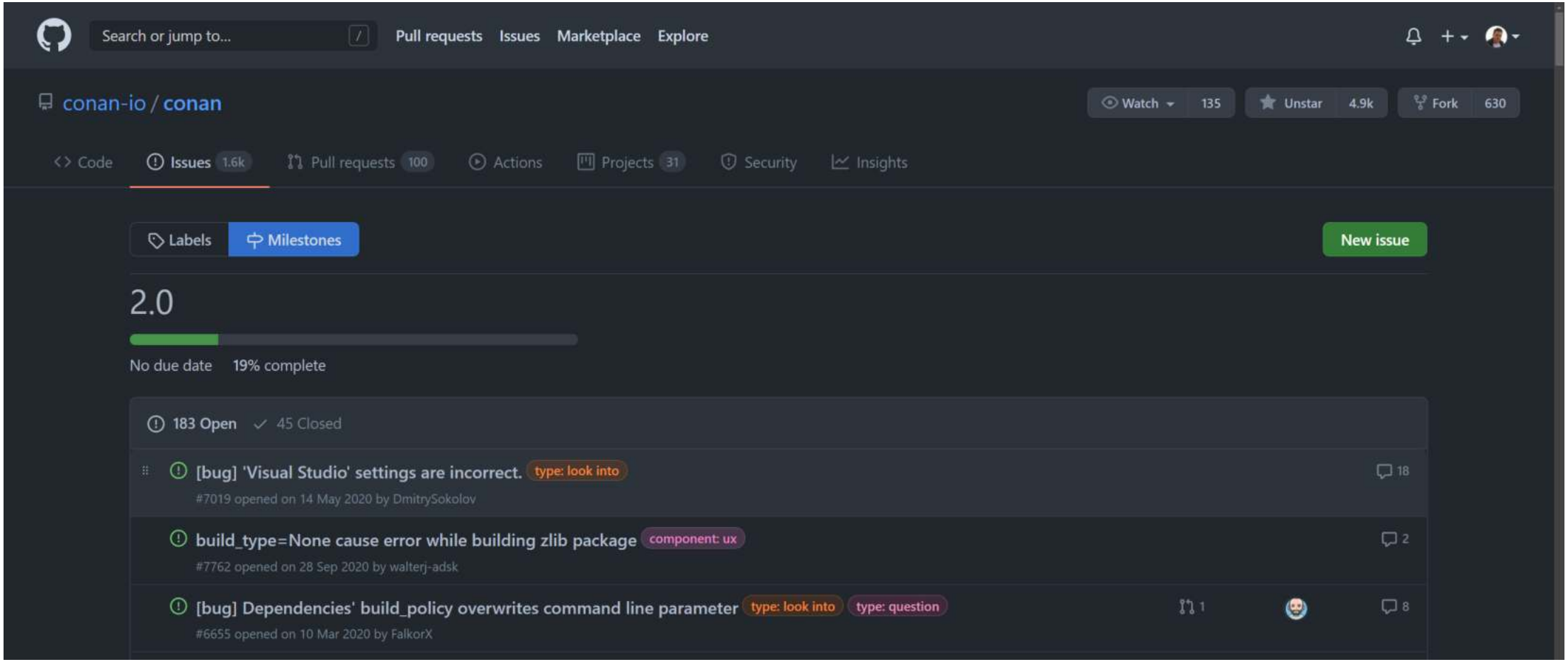
```
def package_info(self):
    self.cpp_info.name = "OpenSSL"
    self.cpp_info.components["crypto"].libs = ["libcrypto"]
    self.cpp_info.components["crypto"].defines = ["DEFINE_CRYPTO=1"]
    self.cpp_info.components["ssl"].includedirs = ["include/headers_ssl"]
    self.cpp_info.components["ssl"].libs = ["libssl"]
    self.cpp_info.components["ssl"].requires = ["crypto"]
```

Unfortunately, this duplicates data from CMake's **`install(TARGETS)`**.

The best resource to find good recipes



Conan 2.0


Search or jump to...
Pull requests
Issues
Marketplace
Explore
conan-io / conan
Watch 135
Unstar 4.9k
Fork 630
Code
Issues 1.6k
Pull requests 100
Actions
Projects 31
Security
Insights
Labels
Milestones
New issue
2.0
No due date
19% complete
183 Open
45 Closed
[bug] 'Visual Studio' settings are incorrect.
type: look into
#7019 opened on 14 May 2020 by DmitrySokolov
18
build_type=None cause error while building zlib package
component: ux
#7762 opened on 28 Sep 2020 by walterj-adsk
2
[bug] Dependencies' build_policy overwrites command line parameter
type: look into
type: question
#6655 opened on 10 Mar 2020 by FalkorX
1
8

Conan 2.0: Compatibility mode

- **`CONAN_V2_MODE`** environment variable
- Good mechanism to **check the recipes for future Conan 2.0 compliance**
- **Raise errors** for deprecated things

Conan 2.0: The most important changes (so far)

- First level setting **`cppstd`** is removed
- Revisions are enabled by default
- GCC >= 5 auto-detected profile will use **`libstdc++11`**
- Forbid access to **`self.cpp_info`** in **`conanfile::package_id()`** method
- Deprecate **`conanfile::config()`** method
- **`default_options`** are required to be a dictionary
- Forbid **`self.settings`** and **`self.options`** in **`conanfile::source()`** method
- Deprecate **`cpp_info.cppflags`** (use **`cpp_info.cxxflags`** instead)
- Deprecate environment variables **`CONAN_USERNAME`** and **`CONAN_CHANNEL`**
- If **`build_type`** or **`compiler`** are not defined when using build helpers Conan will raise an error
- New compiler detection algorithm is used (e.g. when running **`conan profile new <name> --detect`**)

mp-units `conanfile.py`

```
from conans import ConanFile, tools
from conans.tools import Version, check_min_cppstd
from conan.tools.cmake import CMakeToolchain, CMakeDeps, CMake
from conans.errors import ConanInvalidConfiguration
import os, re

required_conan_version = ">=1.33.0"

class UnitsConan(ConanFile):
    name = "mp-units"
    homepage = "https://github.com/mpusz/units"
    description = "Physical Units library for C++"
    topics = ("units", "dimensions", "quantities", "dimensional-analysis", "physical-quantities", "physical-units",
        "system-of-units", "cpp20", "library", "quantity-manipulation")
    license = "MIT"
    url = "https://github.com/mpusz/units"
    settings = "os", "compiler", "build_type", "arch"
    requires = (
        "fmt/7.1.3",
        "gsl-lite/0.38.0"
    )
```

mp-units `conanfile.py`

```
    options = {
        "downcast_mode": ["off", "on", "auto"],
        "build_docs": [True, False]
    }
    default_options = {
        "downcast_mode": "on",
        "build_docs": True
    }
    exports = ["LICENSE.md"]
    exports_sources = ["docs/*", "src/*", "test/*", "cmake/*", "example/*","CMakeLists.txt"]
    generators = "cmake_paths"

    _cmake = None

    @property
    def _run_tests(self):
        return tools.get_env("CONAN_RUN_TESTS", False)

    def set_version(self):
        content = tools.load(os.path.join(self.recipe_folder, "src/CMakeLists.txt"))
        version = re.search(r"project\([^\)]+VERSION (\d+\.\d+\.\d+)[^\)]*\)", content).group(1)
        self.version = version.strip()
```

mp-units conanfile.py

```
    def requirements(self):
        compiler = self.settings.compiler
        if compiler == "clang" and compiler.libcxx == "libc++":
            self.requires("range-v3/0.11.0")

    def build_requirements(self):
        if self._run_tests:
            self.build_requires("catch2/2.13.4")
            self.build_requires("linear_algebra/0.7.0@public-conan/stable")
            if self.options.build_docs:
                self.build_requires("doxygen/1.8.20")

    def validate(self):
        compiler = self.settings.compiler
        version = Version(self.settings.compiler.version)
        if compiler == "gcc":
            if version < "10.0":
                raise ConanInvalidConfiguration("mp-units requires at least g++-10")
        elif compiler == "clang":
            if version < "12":
                raise ConanInvalidConfiguration("mp-units requires at least clang++-12")
        elif compiler == "Visual Studio":
            if version < "16":
                raise ConanInvalidConfiguration("mp-units requires at least Visual Studio 16.9")
        else:
            raise ConanInvalidConfiguration("Unsupported compiler")
        check_min_cppstd(self, "20")
```

mp-units `conanfile.py`

```
    def generate(self):
        tc = CMakeToolchain(self, generator=os.getenv("CONAN_CMAKE_GENERATOR"))
        tc.variables["UNITS_DOWNCAST_MODE"] = str(self.options.downcast_mode).upper()
        tc.variables["UNITS_BUILD_DOCS"] = self.options.build_docs
        tc.generate()
        deps = CMakeDeps(self)
        deps.generate()

    def build(self):
        cmake = CMake(self)
        cmake.configure(source_folder=None if self._run_tests else "src")
        cmake.build()
        if self._run_tests:
            cmake.test(output_on_failure=True)

    def package(self):
        self.copy(pattern="LICENSE.md", dst="licenses")
        cmake = CMake(self)
        cmake.configure(source_folder=None if self._run_tests else "src")
        cmake.install()

    def package_id(self):
        self.info.header_only()
```

mp-units conanfile.py

```
    def package_info(self):
        compiler = self.settings.compiler

        # core
        self.cpp_info.components["core"].requires = ["gsl-lite::gsl-lite"]
        if compiler == "Visual Studio":
            self.cpp_info.components["core"].cxxflags = ["/utf-8"]
        elif compiler == "clang" and compiler.libcxx == "libc++":
            self.cpp_info.components["core"].requires.append("range-v3::range-v3")

        # rest
        self.cpp_info.components["core-io"].requires = ["core"]
        self.cpp_info.components["core-fmt"].requires = ["core", "fmt::fmt"]
        self.cpp_info.components["isq"].requires = ["core"]
        self.cpp_info.components["isq-natural"].requires = ["isq"]
        self.cpp_info.components["si"].requires = ["isq"]
        self.cpp_info.components["si-cgs"].requires = ["si"]
        self.cpp_info.components["si-fps"].requires = ["si"]
        self.cpp_info.components["si-iau"].requires = ["si"]
        self.cpp_info.components["si-imperial"].requires = ["si"]
        self.cpp_info.components["si-international"].requires = ["si"]
        self.cpp_info.components["si-typographic"].requires = ["si"]
        self.cpp_info.components["si-uscs"].requires = ["si"]
        self.cpp_info.components["isq-iec80000"].requires = ["si"]
        self.cpp_info.components["systems"].requires = ["isq", "isq-natural", "si",
            "si-cgs", "si-fps", "si-iau", "si-imperial", "si-international", "si-typographic", "si-uscs", "isq-iec80000"]
```


CAUTION
Programming
is addictive
(and too much fun)