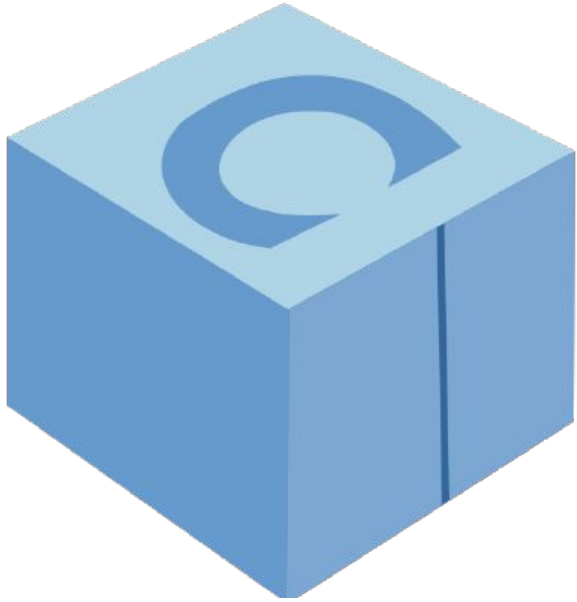




# Conan package manager

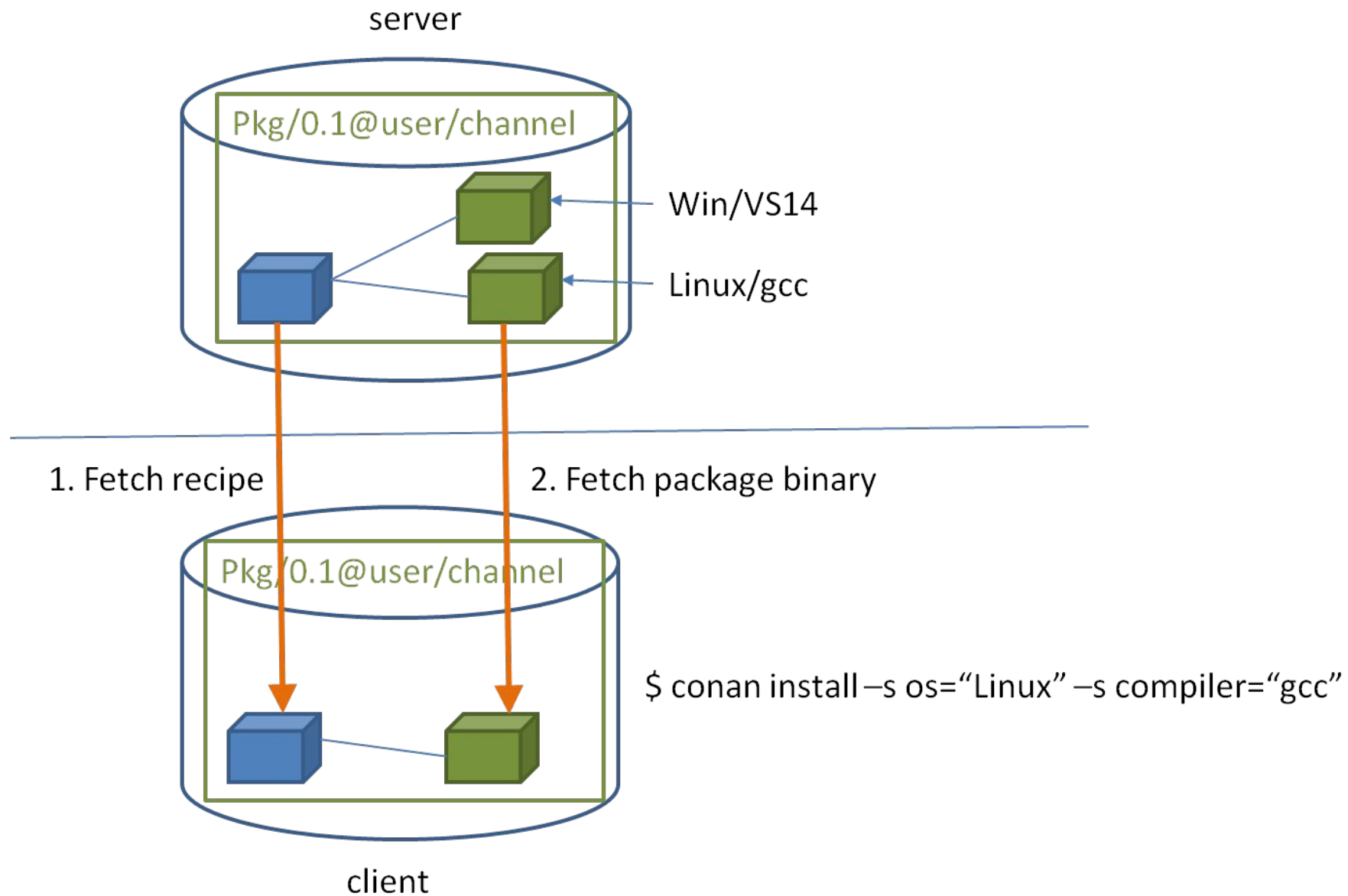
Dariusz Grabowski



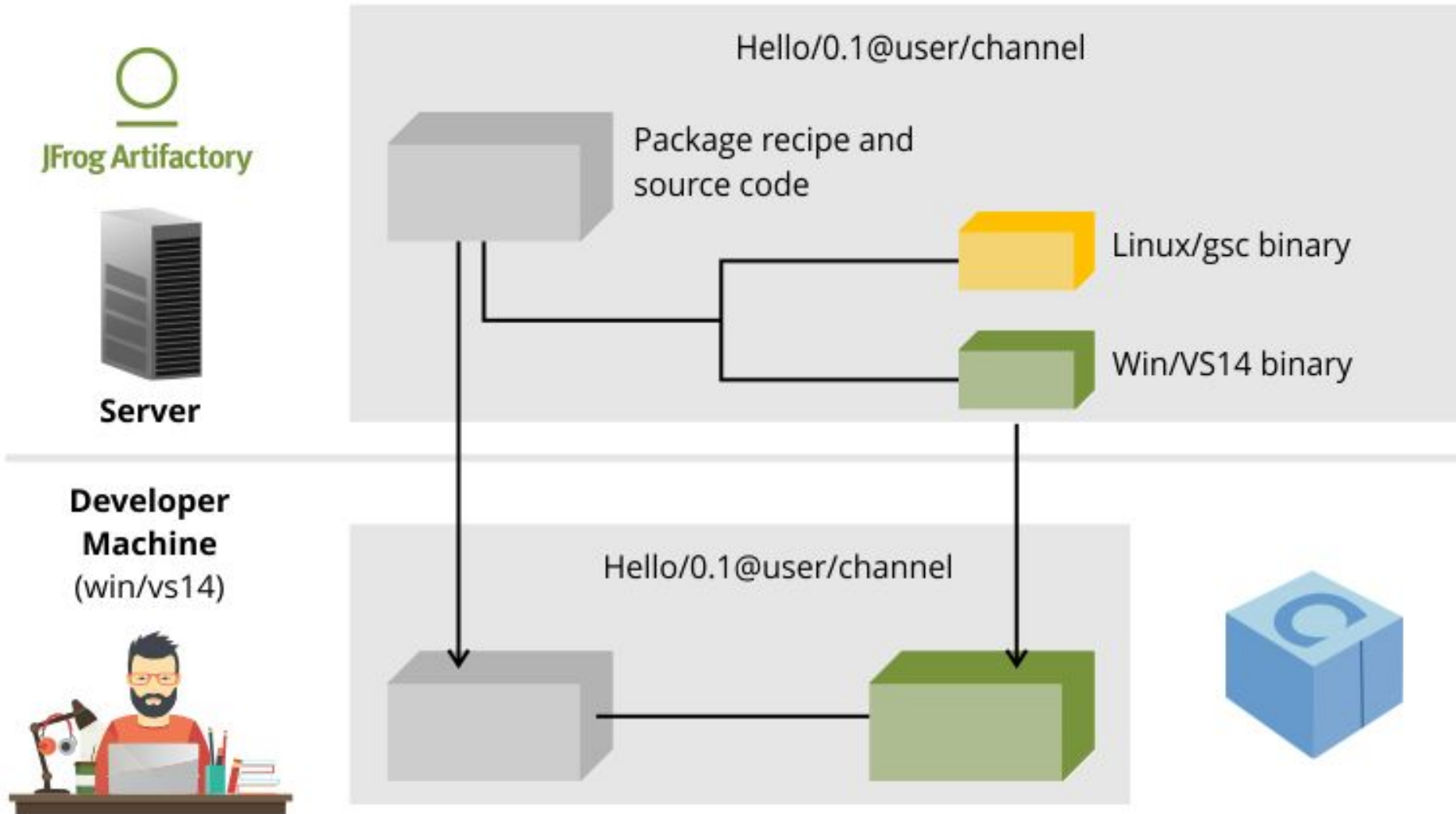
Czym jest conan?

- multiplatformowe narzędzie do zarządzania paczkami C/C++
- wspiera zarządzanie parametrami cmake
- wsparcie procesu crosscompilacji
- umożliwia pobieranie zbudowanych paczek z centralnych repozytoriów
- umożliwia przechowywanie paczek w lokalnym repozytorium
- posiada interfejs do umieszczania zbudowanych paczek np. w Artifactory

# Conan



# Conan



# Conan receptura

- conanfile.py opisuje sposób budowania paczki oraz jej parametry
- metody odpowiadają kolejnym krokom:
  - source
  - build
  - package
- możliwe jest zdefiniowanie parametrów kompilacji (settings) oraz opcji (options)
  - zmiana *settings* powoduje utworzenie nowej paczki
  - a zmiana opcji już nie
- metoda `package_info` wskazuje które biblioteki/pliki nagłówkowe mają być dodane do modułu `cmake` opisującego budowaną paczkę

```
class LibAConan(ConanFile):
    name = "libB"
    version = "0.1"
    license = "MIT"
    author = "pkgAuthor pkgAuthor@mail.com"
    url = "www.google.com"
    description = "B is a powerfull cpp library"
    settings = "os", "compiler", "build_type", "arch"
    options = {"shared": [True, False], "runtests": [True, False], "coverage":
[True, False]}
    default_options = {"shared": True, "runtests": True, "coverage": True }
    generators = "cmake_find_package"
    exports_sources = ["software/*"]
    requires = "gtest/1.8.1@bincrafters/stable", "libC/0.1@ci_poc/testing"

    def configure_cmake(self):
        cmake = CMake(self)
        if self.options.coverage:
            definitions["CODE_COVERAGE"] = True
        cmake.configure(source_folder="software", defs=definitions)
        return cmake

    def build(self):
        cmake = self.configure_cmake()
        cmake.build()
        if self.options.runtests:
            cmake.test()

    def package(self):
        cmake = self.configure_cmake()
        cmake.install()

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

Conanfile.txt to prosty sposób na opisanie zależności pomiędzy paczkami. Można w nim zawrzeć sekcje:

- wymagane paczki
- generator - z jakiego narzędzia korzystał będzie konsument
- ustawienia dla pobieranych paczek

Ta wersja receptury jest bardzo prosta ale nakłada duże ograniczenia.

```
[requires]
Poco/1.9.0@pocoproject/stable
zlib/1.2.11@conan/stable
```

```
[generators]
cmake
```

```
[options]
Poco:shared=True
OpenSSL:shared=True
```

# conan install

```
cmd: conan install .. \  
-s compiler=gcc \  
-o shared=True \  
-o gtest:shared=False
```

1. Sprawdzana jest konfiguracja (plik profile + przekazane settings + options)
2. Sprawdzane i instalowane są zależności, jeśli nie istnieją w lokalnym cache - pobierane z zewnątrz
3. Generowany jest plik conanbuildinfo.cmake (możemy go później zaincludować w CMakeLists.txt)

```
Configuration:  
[settings]  
arch=x86_64  
arch_build=x86_64  
build_type=Debug  
compiler=gcc  
compiler.libcxx=libstdc++11  
compiler.version=5.4  
os=Linux  
os_build=Linux  
[options]  
[build_requires]  
[env]  
  
conanfile.py (connectionManager/0.1): Installing package  
Requirements  
  common/0.1@ci/training from local cache - Cache  
  gtest/1.8.1@binrafters/stable from 'conan-center' - Cache  
Packages  
  common/0.1@ci/training:25e4462452b2782a168076cf14823c8bc43713fe - Cache  
  gtest/1.8.1@binrafters/stable:b98b0d9db9614a411feb0a307cd4fbe7cf833fa1 - Cache  
  
common/0.1@ci/training: Already installed!  
gtest/1.8.1@binrafters/stable: Already installed!  
conanfile.py (connectionManager/0.1): Generator cmake created conanbuildinfo.cmake  
conanfile.py (connectionManager/0.1): Generator txt created conanbuildinfo.txt  
conanfile.py (connectionManager/0.1): Generated conaninfo.txt  
conanfile.py (connectionManager/0.1): Generated graphinfo
```



cmd: conan build ..

- Do pliku CMakeLists.txt należy dołączyć wygenerowany plik conanbuildinfo
- wywołanie `conan_basic_setup()` powoduje:
  - sprawdzenie wersji kompilatora
  - przekazanie definicji
  - przekazanie ustawień kompilacji (np. SHARED/STATIC)
  - ustawienie ścieżek do modułów
  - ustawienie standardu języka + wersji api

```
# CMakeLists.txt
cmake_minimum_required(VERSION 3.15)
project(myProject)

include(${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
conan_basic_setup(TARGETS)

find_package(Threads)
add_library(...)
```

# generator “cmake”

- Generator dający największą swobodę działania - zgodny z “modern approach”
- wywołanie `conan_basic_setup()` powoduje:
  - sprawdzenie wersji kompilatora
  - przekazanie definicji
  - przekazanie ustawień kompilacji (np. SHARED/STATIC)
  - ustawienie ścieżek do modułów
  - ustawienie standardu języka + wersji api

## conanfile.txt

```
[requires]
libcurl/7.52.1@bincrafters/stable

[generators]
cmake

[options]
libcurl:with_openssl=True
```

## CMakeLists.txt

```
project(myapp)
cmake_minimum_required(VERSION 3.1)

include(${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
conan_basic_setup(TARGETS)

add_executable(myapp main.cpp)
target_link_libraries(myapp CONAN_PKG::libcurl)
```

# generator “cmake\_paths”

```
# conanfile.txt

[requires]
libcurl/7.52.1@bincrafters/stable

[generators]
cmake_paths

[options]
libcurl:with_openssl=True
```

```
# CMakeLists.txt
PROJECT(myapp)
cmake_minimum_required(VERSION 3.1)

ADD_EXECUTABLE(myapp main.cpp)

find_package(ZLIB)
find_package(OpenSSL) # need to add
find_package(CURL) # need to add

include_directories(${ZLIB_INCLUDE_DIRS}
${OPENSSL_INCLUDE_DIRS} ${CURL_INCLUDE_DIRS})
target_link_libraries(myapp ${CURL_LIBRARIES}
${OPENSSL_LIBRARIES} ${ZLIB_LIBRARIES})
```

```
$ cmake .. -DCMAKE_BUILD_TYPE=Release -DCMAKE_TOOLCHAIN_FILE=./conan_paths.cmake
$ cmake --build .
$ ./bin/myapp
```

## generator “cmake\_find\_package”

```
# conanfile.txt

[requires]
libcurl/7.52.1@bincrafters/stable

[generators]
cmake_find_package

[options]
libcurl:with_openssl=True
```

```
# CMakeLists.txt
project(myapp)
cmake_minimum_required(VERSION 3.1)
set(CMAKE_MODULE_PATH ${CMAKE_BINARY_DIR})
message(${CMAKE_BINARY_DIR})

add_executable(myapp main.cpp)
find_package(libcurl)
target_link_libraries(myapp libcurl::libcurl)
```

```
$ cmake .. -DCMAKE_BUILD_TYPE=Release
$ cmake --build .
$ ./bin/myapp
```

# Conan profiles

- globalne dla wszystkich projektów
- można w nich zdefiniować wersję kompilatora, architekturę, typ budowania (Build/Release itd.), uruchamianie testów i inne
- Przy zmianie kompilatora/sposobu budowania nie ma konieczności modyfikowania plików CMake
- Dla profili części wspólne można wydzielić do osobnego pliku a później includować

```
[settings]
os=Linux
os build=Linux
arch=x86_64
arch build=x86_64
compiler=gcc
compiler.version=7
compiler.libcxx=libstdc++11
build type=Debug
[options]
[build_requires]
[env]
```

---

```
[settings]
zlib:compiler=clang
zlib:compiler.version=3.5
zlib:compiler.libcxx=libstdc++11
compiler=gcc
compiler.version=4.9
compiler.libcxx=libstdc++11

[env]
zlib:CC=/usr/bin/clang
zlib:CXX=/usr/bin/clang++
```

```
[settings]
os=Linux
os build=Linux
arch=x86_64
arch build=x86_64
compiler=gcc
compiler.version=7
compiler.libcxx=libstdc++11
build type=Release
[options]
[build_requires]
[env]
```

---

```
[settings]
os=Windows
arch=x86_64
compiler=Visual Studio
compiler.version=15
build_type=Release
[options]
[build_requires]
[env]
```

```
conan create . demo/testing --profile clang
```

# Conan profiles

## zlib/1.2.11@conan/stable

	x86 Debug shared=False	x86 Debug shared=True	x86 Release shared=False	x86 Release shared=True	x86_64 Debug shared=False	x86_64 Debug shared=True	x86_64 Release shared=False	x86_64 Release shared=True
Linux clang 3.9								
Linux clang 4.0								
Linux gcc 4.9								
Linux gcc 5.4								
Linux gcc 6.3								
Macos apple-clang 7.3								
Macos apple-clang 8.0								
Macos apple-clang 8.1								
Windows Visual Studio 10 (MD)								
Windows Visual Studio 10 (MDd)								
Windows Visual Studio 10 (MT)								
Windows Visual Studio 10 (MTd)								
Windows Visual Studio 12 (MD)								
Windows Visual Studio 12 (MDd)								
Windows Visual Studio 12 (MT)								
Windows Visual Studio 12 (MTd)								
Windows Visual Studio 14 (MD)								
Windows Visual Studio 14 (MDd)								
Windows Visual Studio 14 (MT)								
Windows Visual Studio 14 (MTd)								
Windows Visual Studio 15 (MD)								
Windows Visual Studio 15 (MDd)								
Windows Visual Studio 15 (MT)								
Windows Visual Studio 15 (MTd)								
Windows gcc 4.9 (dwarf2) (posix)								
Windows gcc 4.9 (seh) (posix)								
Windows gcc 4.9 (sjlj) (posix)								

Selected:

52365c918e417dff048f3ad367c434eb2c362d08

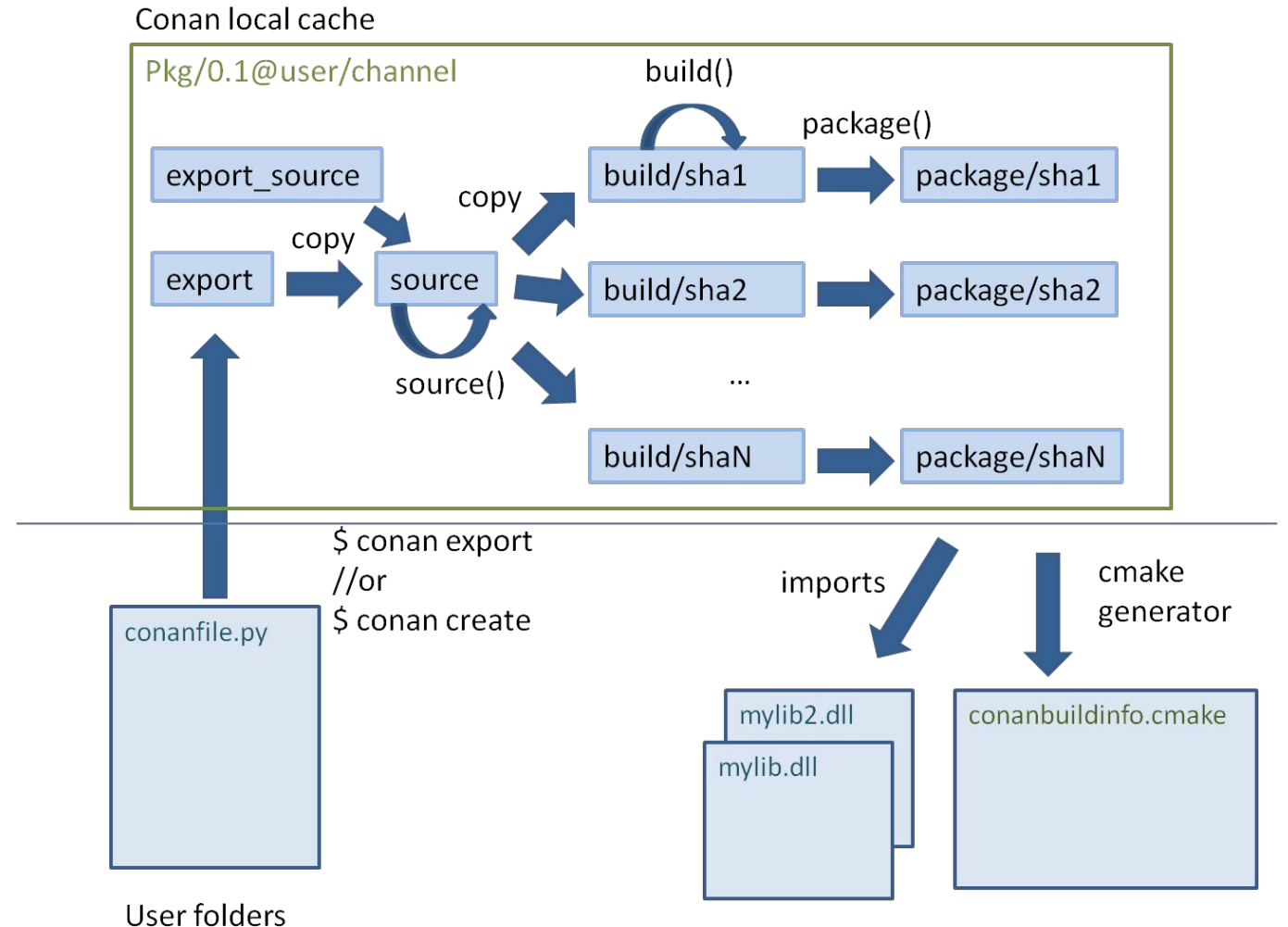
Legend

	Outdated from recipe
	Updated
	Non existing

# Conan magazynowanie




Każdy folder komponentu w lokalnym cache zawiera:


- **export**: folder dla receptury
- **export\_source**: tutaj znajdują się wyeksportowane źródła paczki jeśli przechowywane są razem z recepturą
- **source**: katalog na źródła do budowania (pobrane z repozytorium + export\_source + export)
- **build**: katalog na budowanie źródeł. Podkatalogi dla każdej z konfiguracji.
- **package**: katalog na zbudowane paczki. Podkatalogi dla każdej z konfiguracji.





# Conan + artifactory

 **JFrog Artifactory**  Welcome, admin  [Help](#)

 **Build Browser**

All Builds > TestPackageGeneric > 13

Build #13

General Build Info Published Modules Environment Issues Licenses Diff >>

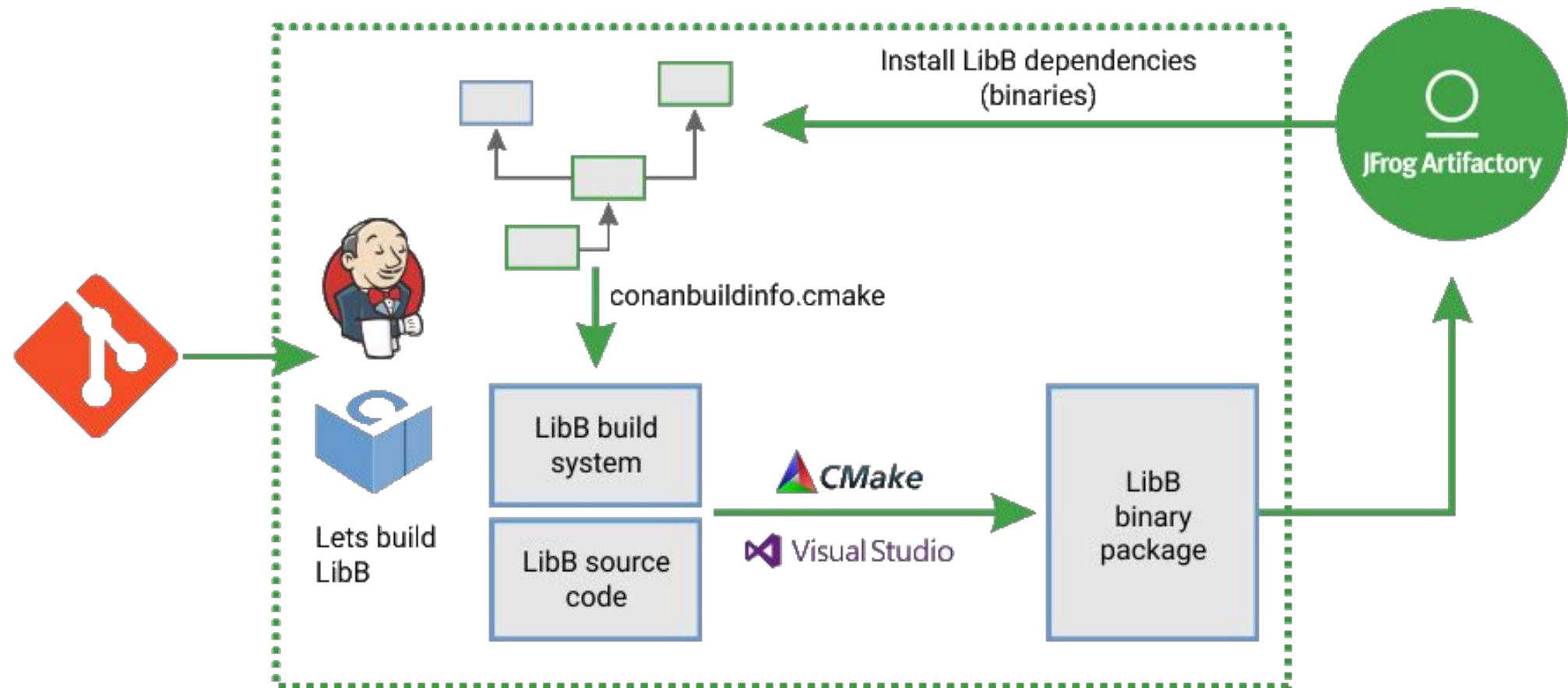
4 Modules

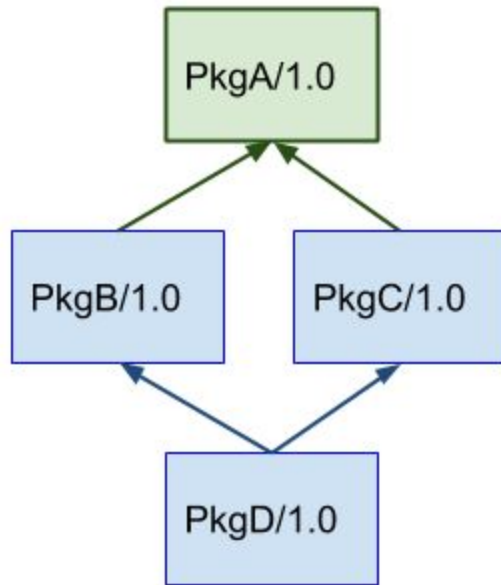
< Page 1 of 1 >

Module ID	Number Of Artifacts	Number Of Dependencies
<a href="#">libcurl/7.50.3@lasote/testing</a>	3	3
<a href="#">libcurl/7.50.3@lasote/testing:0f7474d0a0daebaf9f2ff9db477cf06ae5f7ed1</a>	3	3
<a href="#">zlib/1.2.8@lasote/stable</a>	3	0
<a href="#">zlib/1.2.8@lasote/stable:09512ff863f37e98ed748eadd9c6df3e4ea424a8</a>	3	0

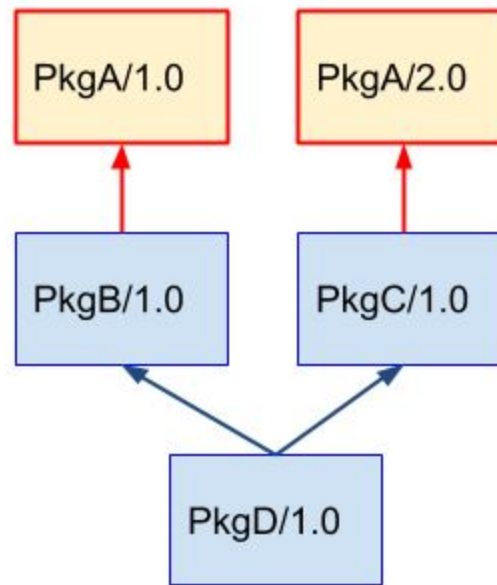


# Conan

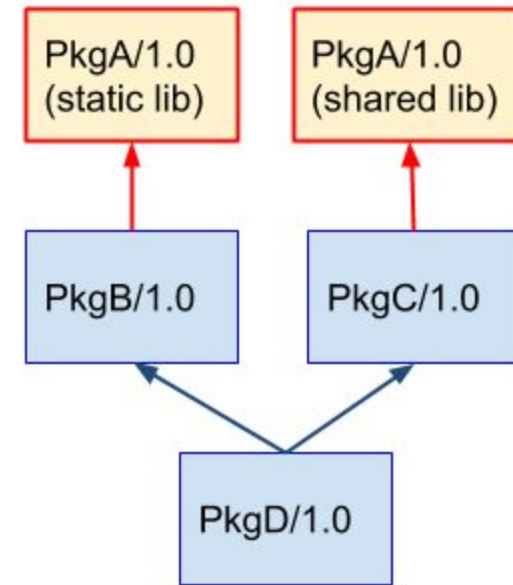




“Diamond” in the dependency graph.



Error: Version conflict.

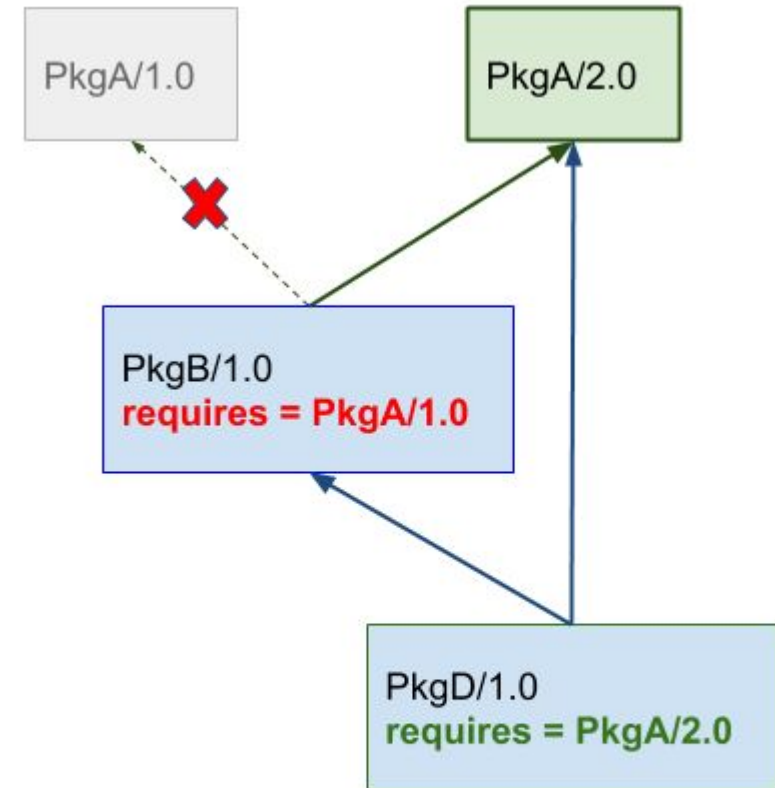


Error: Configuration conflict

# Wersjonowanie

Paczki będące niżej w hierarchii mają priorytet co do wyboru wersji paczki - zależności mogą zostać automatycznie nadpisane.

Można to zachowanie zmodyfikować poprzez parametr `CONAN_ERROR_ON_OVERRIDE` w pliku konfiguracyjnym conana lub poprzez zmienną środowiskową.



PkgD/1.0 defines a requirement to PkgA/2.0, overriding PkgB definition pointing to PkgA/1.0

# Wersjonowanie

```
[>1.1 <2.1]      # w zakresie od 1.1 do 2.1
[2.8]             # precyzyjna wersja, to samo co =2.8
[>1.1 || 0.8]     # jedno lub drugie
[1.2.7 || >=1.2.9 <2.0.0] # Możliwe wersje to 1.2.7, 1.2.9, i 1.4.6, ale 1.2.8
or 2.0.0 już nie
```

```
Pkg/[>1.0 <1.8]@user/stable # wersja jako numer
Pkg/1.1@user/stable         # wersja jako string
```

- Jeśli nie użyjemy nawiasów [ ] wersja paczki traktowana jest jako string
- Nie wszystkie paczki wspierają ten sposób wersjonowania - przykład OpenSSL 1.0.2n
- Aby pobrać najnowszą wersję paczek należy użyć polecenia:  
conan install .. --update

# Testowanie paczek

- Podczas wywoływania polecenia `conan create` sprawdzane jest istnienie katalogu `test_package`
- Jeśli istnieje - w podkatalogu zostanie utworzony folder `build` a projekt `cmake` zostanie zbudowany
- Projekt ten powinien sprawdzać poprawność budowania paczki
- Należy zwrócić uwagę, że nie powinny to być unit testy lub integracyjne - te warto umieścić w `src`
- szablon można utworzyć poprzez:  
`conan new -s -t myPkg/1.0@user/stable`

```
├── conanfile.py
├── src
│   ├── CMakeLists.txt
│   ├── hello.cpp
│   └── hello.h
└── test_package
    ├── CMakeLists.txt
    ├── conanfile.py
    └── example.cpp
```



Pytania?

# Zadanie

1. Zbudować paczkę Hello

`conan new Hello/0.1`

2. Stworzyć profil dla clanga (zainstalować jeśli brakuje w systemie)
3. W źródłach poprzedniego projektu (modern cmake) skonsumować Hello
4. Zbudować projekt paczkę z profilem clang i gcc.
5. Jeśli to konieczne - poprawić cmakelisty z poprzedniego projektu
6. Sprawdzić istnienie tych paczek w lokalnym cache

Wymagania/ograniczenia:

1. Paczka powinna zawierać wszystkie biblioteki i pliki nagłówkowe.
2. Projekt powinien zawierać katalog `test_package` z odpowiednim testem.