conan.io

•••

C/C++ package manager

cmake

cmake

cmake is cross-platform tool managing the build process of software using a compiler independent method. It supports directory hierarchies and applications that depend on multiple libraries. It is used in conjunction with native build environments such as make, Apple's Xcode, and Microsoft Visual Studio.

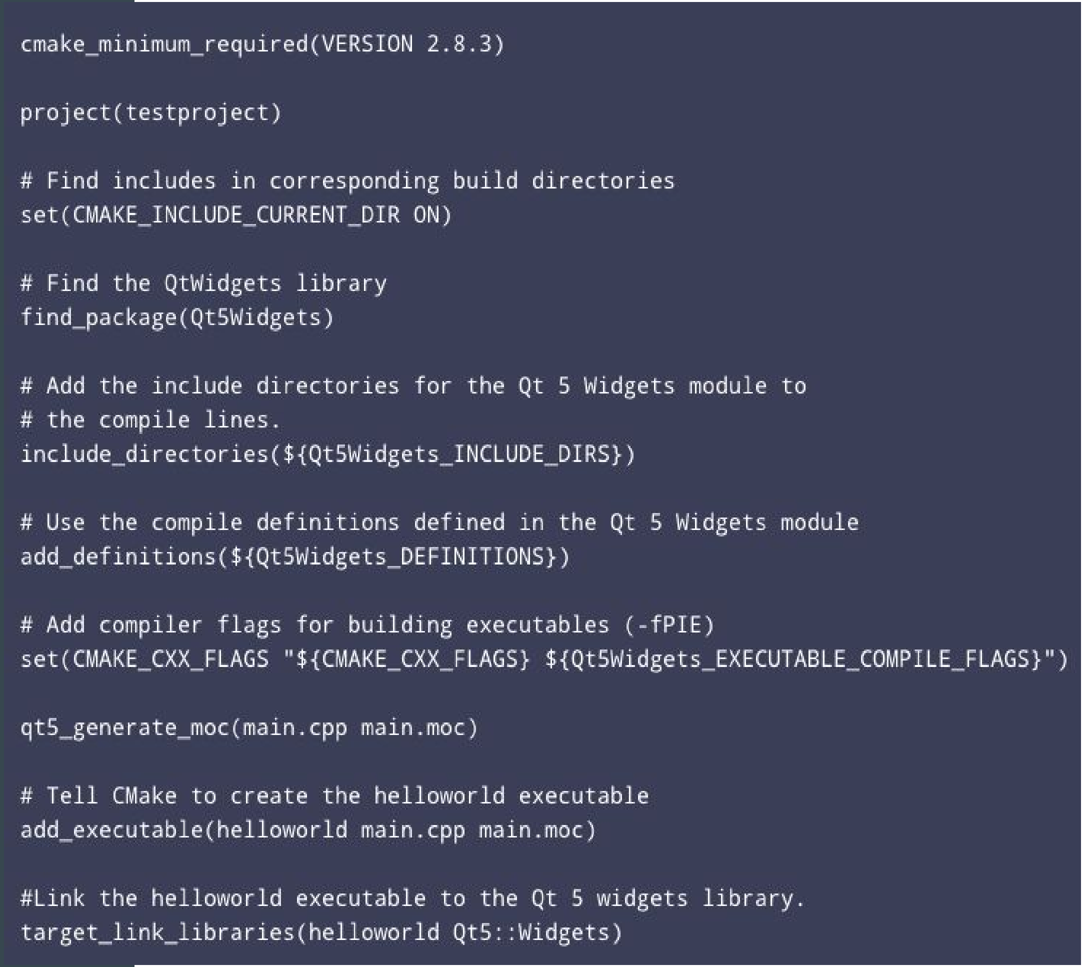
It has minimal dependencies, requiring only a C++ compiler on its own build system.

cmake modules

- cmake-buildsystem
  - Targets: executables, libraries.
- cmake-generators
  - Input files for build systems: cmd line build tool (makefiles), IDEs (Visual Studio, XCode).
- cmake-packages
  - Package/module navigation.
  - Results is set imported targets or set built-relevant variables.

cmake-package

Qt5 example



```
cmake_minimum_required(VERSION 2.8.3)

project(testproject)

# Find includes in corresponding build directories
set(CMAKE_INCLUDE_CURRENT_DIR ON)

# Find the QtWidgets library
find_package(Qt5Widgets)

# Add the include directories for the Qt 5 Widgets module to
# the compile lines.
include_directories(${Qt5Widgets_INCLUDE_DIRS})

# Use the compile definitions defined in the Qt 5 Widgets module
add_definitions(${Qt5Widgets_DEFINITIONS})

# Add compiler flags for building executables (-fPIE)
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${Qt5Widgets_EXECUTABLE_COMPILE_FLAGS}")

qt5_generate_moc(main.cpp main.moc)

# Tell CMake to create the helloworld executable
add_executable(helloworld main.cpp main.moc)

#Link the helloworld executable to the Qt 5 widgets library.
target_link_libraries(helloworld Qt5::Widgets)
```

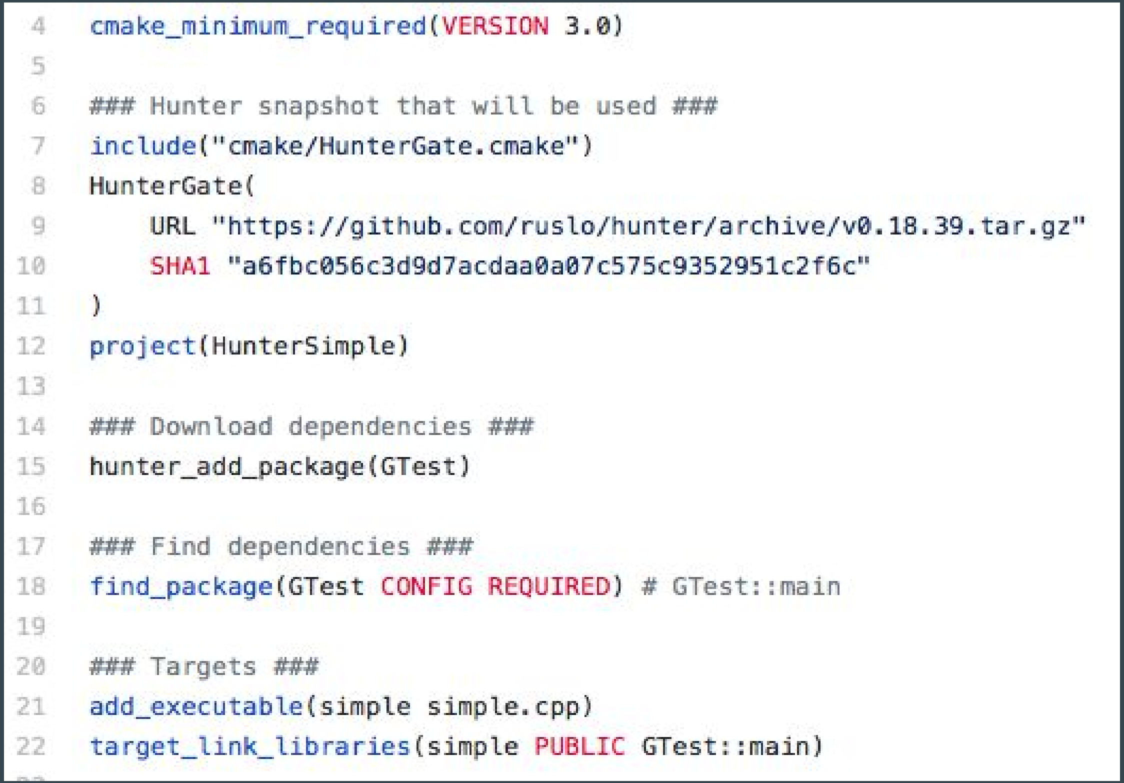
Hunter

Hunter

Hunter is a CMake-driven cross-platform package manager for C++. It is designed to manage packages with CMake build system under the hood and existing CMake packages can be quite easily integrated into system, but non-CMake packages are also supported too using custom templates.

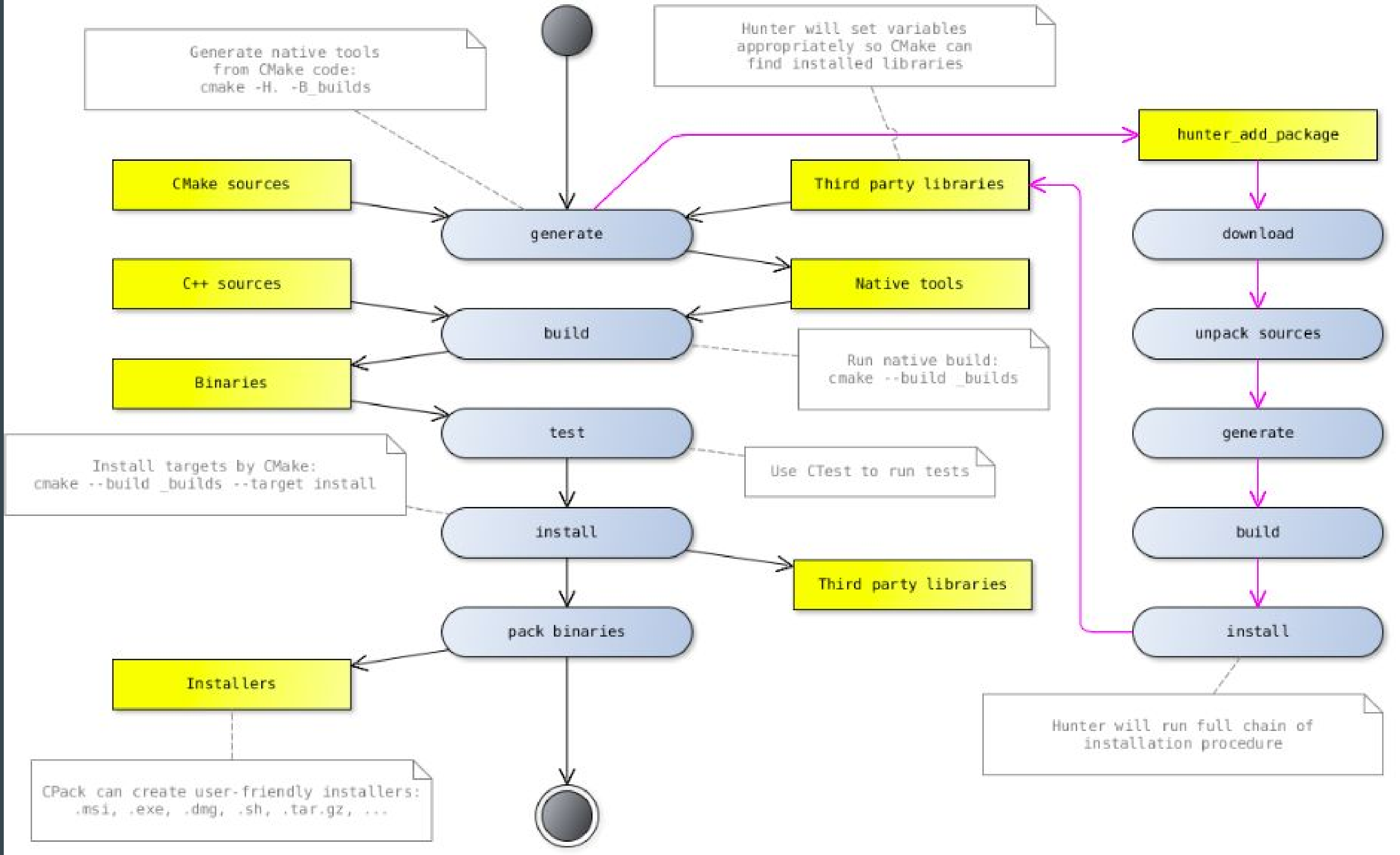
The Hunter client is a collection of CMake-only modules (i.e. it's not a binary like apt-get or script like brew) so it supports out-of-the-box all platforms/generators/IDEs which CMake can handle, like Visual Studio, Xcode, QtCreator, NMake, Ninja, Cygwin or MinGW. It works fine with CMake-GUI too.

Hunter example



```
cmake_minimum_required(VERSION 3.0)

### Hunter snapshot that will be used ###
include("cmake/HunterGate.cmake")
HunterGate(
    URL "https://github.com/ruslo/hunter/archive/v0.18.39.tar.gz"
    SHA1 "a6fbc056c3d9d7acdaa0a07c575c9352951c2f6c"
)
project(HunterSimple)

### Download dependencies ###
hunter_add_package(GTest)

### Find dependencies ###
find_package(GTest CONFIG REQUIRED) # GTest::main

### Targets ###
add_executable(simple simple.cpp)
target_link_libraries(simple PUBLIC GTest::main)
```

Hunter flow


Generate native tools
from CMake code:
cmake -H. -B_builds
Hunter will set variables
appropriately so CMake can
find installed libraries
hunter_add_package
CMake sources
Third party libraries
generate
download
C++ sources
Native tools
build
unpack sources
Run native build:
cmake --build _builds
Binaries
test
generate
Install targets by CMake:
cmake --build _builds --target install
Use CTest to run tests
install
build
Third party libraries
pack binaries
install
Installers
Hunter will run full chain of
installation procedure
CPack can create user-friendly installers:
.msi, .exe, .dmg, .sh, .tar.gz, ...

Maven

Maven

Apache Maven is a software project management and comprehension tool.

Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

Maven NAR plugin

- Compilation on different architectures using different compilers.
- Output wrapped in (.nar) file - Native ARchive file.
- Output types:
  - platform independent output,
  - platform specific, depending on Architecture, Operation system, Linker (AOL).

cmake-maven-project

A Maven project for the CMake build system. It can be used by including it as a plugin within your Maven project's pom.xml file.

conan.io

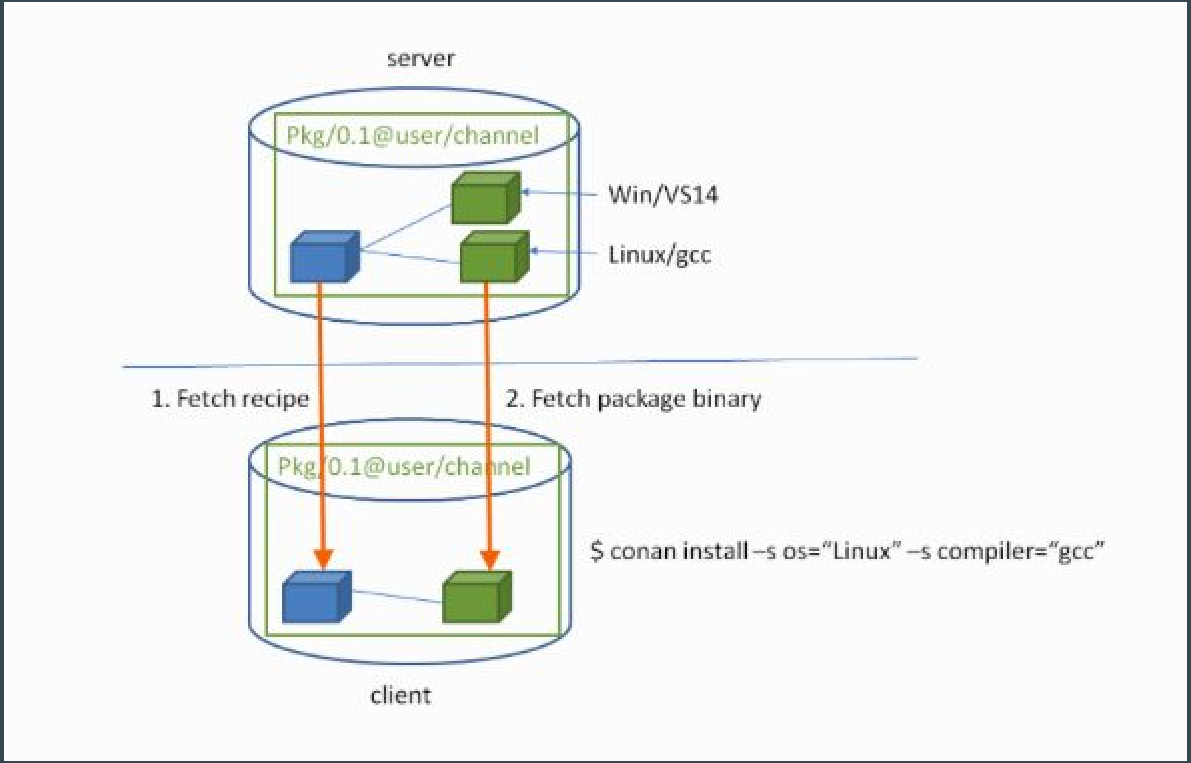
conan.io

Conan is a portable python-based, package manager, intended for C and C++ developers, but it is able to manage builds from source, dependencies, and precompiled binaries for any language.

conan.io features

- De-centralized
  - local cache, remote conan_server, conan-transit, conan-center.
- Binaries and source code
  - Download binaries or build from sources.
- Dependency control
  - Dependency config, overriding, conflict resolution.
- JFrog artifact repo compatibility
  - Use artificatory to host your conan packages.

conan.io - binary management



server

Pkg/0.1@user/channel

Win/VS14

Linux/gcc

1. Fetch recipe

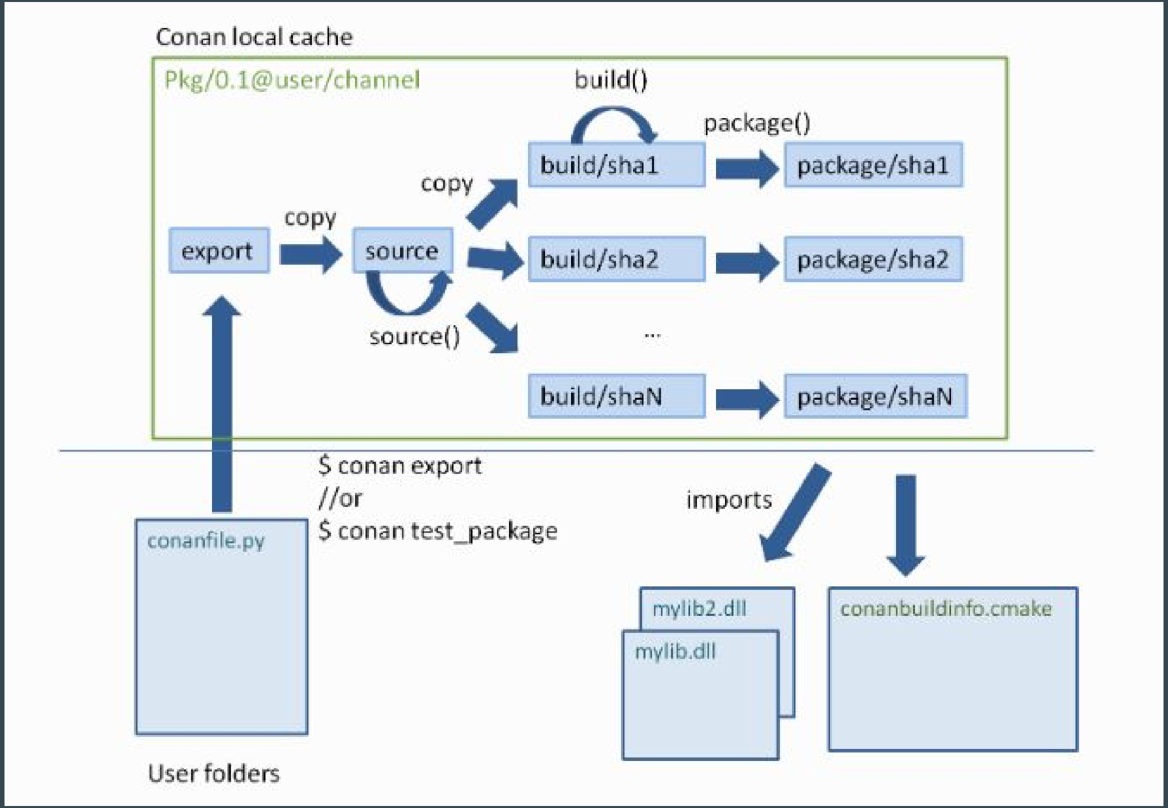
2. Fetch package binary

Pkg/0.1@user/channel

```
$ conan install –s os="Linux" –s compiler="gcc"
```

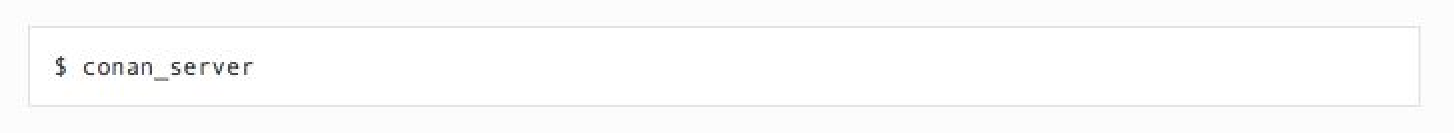
client

conan.io - package in local cache



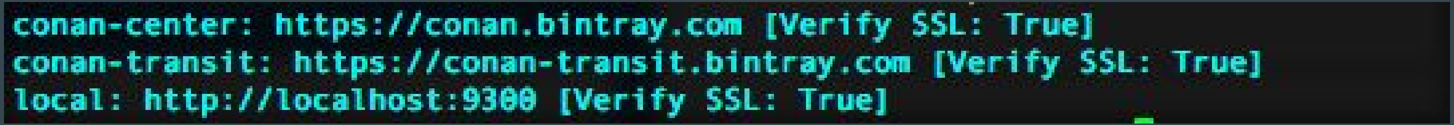
conan_server

- Included in conan install package.



```
$ conan_server
```

- Conan remote server list:



```
conan-center: https://conan.bintray.com [Verify SSL: True]
conan-transit: https://conan-transit.bintray.com [Verify SSL: True]
local: http://localhost:9300 [Verify SSL: True]
```

conan.io - uploading packages

- If you are just evaluating conan and don’t want to register an account on conan.io, it is very simple to run a conan server.

- If you want to upload your packages to the conan.io server, you have to register an account first at conan.io site.

The end