Packaging C/C++ dependencies with Conan
First things first

@theodelrieu

Tanker.io

Conan contributor
First things first

@theodelrieu

Tanker.io

Conan contributor
You said Tanker?

End-to-End encryption SDK

Available in Javascript

Soon™ on Android/iOS
“Starting a new project is awesome!”

“We can have complete control of our stack!”

“Hum... which language shall we use?”

Steve the Intern
A subject of discord
Decisions, decision

Our key conditions:

- Write once, run everywhere
- High performance
- Good dependency management
Decisions, decisions

Our key conditions:

- Write once, run everywhere
- High performance
- Good dependency management
Wait, w00t?!

‘C++’ and ‘good dependency management’ in the same slide...
‘C++’ and ‘good dependency management’ in the same slide...

Usual C++ dependencies:
Behold, Conan!

Decentralized package manager

Open Source

Python
Conan 101: Package Management

Using Conan to cross-build for Android
Conan 101: Creating packages

$ conan create

Run recipe

Store package locally
Conan 101: Creating packages

$ conan create

Run recipe

Store package locally

$ conan upload

Find local package

Upload to server
```python
from conans import ConanFile

class ArithmeticConan(ConanFile):
    name = "arithmetic"
    version = "0.1"
```
from conans import ConanFile

class ArithmeticConan(ConanFile):
    name = "arithmetic"
    version = "0.1"
    settings = "os", "arch", "build_type", "compiler"
from conans import ConanFile

class ArithmeticConan(ConanFile):
    name = "arithmetic"
    version = "0.1"
    settings = "os", "arch", "build_type", "compiler"

    def source(self):
        url = "https://github.com/theodelrieu/FOSDEM2018-arithmetic"
        self.run("git clone %s arithmetic" % url)
from conans import ConanFile

class ArithmeticConan(ConanFile):
    def build(self):
        cmake = CMake(self)
        cmake.configure(source_dir="arithmetic")
        cmake.build()
        cmake.install()
from conans import ConanFile

class ArithmeticConan(ConanFile):
    def build(self):
        cmake = CMake(self)
        cmake.configure(source_dir="arithmetic")
        cmake.build()
        cmake.install()

    def package_info(self):
        self.cpp_info.libs = ["arithmetic"]
Conan 101: Consuming packages

$ conan install

Run recipe

Fetch packages

Generate build info
Conan 101: Consuming packages

conanfile.txt

[requires]
arithmetic/0.1@theo/stable
Conan 101: Consuming packages

conanfile.txt

[requires]
arithmetic/0.1@theo/stable

[generators]
cmake
CMakeLists.txt before Conan

cmake_minimum_required(VERSION 3.0)
project(Calculator)

find_package(Arithmetic)
add_executable(calculator src/main.cpp)
target_link_libraries(calculator Arithmetic::Arithmetic)
CMakeLists.txt
after Conan

cmake_minimum_required(VERSION 3.0)
project(Calculator)
include(${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
conan_basic_setup()
find_package(Arithmetic)
add_executable(calculator src/main.cpp)
target_link_libraries(calculator Arithmetic::Arithmetic)
CMakeLists.txt

after Conan, without find_package

cmake_minimum_required(VERSION 3.0)
project(Calculator)
include(${CMAKE_BINARY_DIR}/conanbuildinfo.cmake)
conan_basic_setup(TARGETS)

add_executable(calculator src/main.cpp)
target_link_libraries(calculator CONAN_PKG::arithmetic)
Conan 101: Change settings

// using older GCC version
$ conan create . theo/stable -s compiler.version=6

// clang 5, new GCC ABI, Debug build. It gets hairy...
$ conan create . theo/stable -s compiler=clang
  -s build_type=Debug -s compiler.version=5.0
  -s compiler.libcxx=libstdc++11
Conan 101: Profiles

Profiles are a solution

# Generated by default (on my machine)
[settings]
os=Linux
arch=x86_64
compiler=gcc
compiler.version=7
compiler.libcxx=libstdc++  # New ABI: libstdc++11
build_type=Release
Conan 101: Profiles

New profile: clang5-debug

```
[settings]
os=Linux
arch=x86_64
compiler=clang
compiler.version=5.0
compiler.libcxx=libstdc++11
build_type=Debug
```
Conan 101: Profiles

Manually specifying settings is still possible:

```
$ conan create . theo/stable --profile clang5-debug
```

```
$ conan create . theo/stable -pr clang5-debug -s ...
```
Conan 101: Package Management

Using Conan to cross-build for Android
Conan & Android

Prerequisites:

- Android NDK
- Standalone Android Toolchain
- New Conan Profile
Conan & Android: Build requirements

**RECIPE**

```
self.env_info.FOO="bar"
```
Conan & Android: Build requirements

```
self.env_info.FOO="bar"
```

- PackageA
- PackageB

**RECIPE**

```
buildRequires
```

Made by Tanker.io team
Conan & Android: Build requirements

**Recipe**

```python
self.env_info.FOO = "bar"
```

**Env**

```bash
export FOO = "bar"
```

Diagram:

- PackageA
- PackageB

**build_requires** from PackageB to PackageA.
Conan & Android: Build requirements

**Package A**

**Recipe**

self.env_info.FOO="bar"

**Recipe**

self.cpp_info.sysroot=/bar

**Recipe**

self.deps_cpp_info.sysroot=/bar

**Package B**

**Env**

export FOO="bar"
Conan & Android: Setting up the Toolchain

Android NDK \text{buildRequires} \text{Standalone Android Toolchain}
Conan & Android: Setting up the Toolchain

- Android NDK
  - build_requires
  - Standalone Android Toolchain
  - build_requires
  - Arithmetic
Android profile

[settings]
os=Android
arch=armv8
os.api_level=21
compiler=clang
compiler.version=5
compiler.libcxx=libc++
build_type=Release
[settings]
os=Android
arch=armv8
os.api_level=21
compiler=clang
compiler.version=5
compiler.libcxx=libc++
build_type=Release
os_build=Linux
arch_build=x86_64
from conans import ConanFile, tools
from os import path, unlink

class AndroidNDKConan(ConanFile):
    name = "android-ndk"
    version = "r16"
from conans import ConanFile, tools
from os import path, unlink

class AndroidNDKConan(ConanFile):
    name = "android-ndk"
    version = "r16"
    settings = "os_build", "arch_build"
from conans import ConanFile, tools
from os import path, unlink

class AndroidNDKConan(ConanFile):
    name = "android-ndk"
    version = "r16"
    settings = "os_build", "arch_build"

    def source(self):
        tools.download(url, NDK_URL)
        tools.unzip("ndk.zip", keep_permissions=True)
        os.unlink("ndk.zip")
from conans import ConanFile, tools
from os import path, unlink

class AndroidNDKConan(ConanFile):
    def package(self):
        self.copy("*", src="android-ndk-r16")
from conans import ConanFile, tools
from os import path, unlink

class AndroidNDKConan(ConanFile):
    def package(self):
        self.copy("*", src="android-ndk-r16")

    def package_info(self):
        tools_folder = path.join(self.package_folder, "build/tools")
        self.env_info.PATH.append(tools_folder)
from conans import ConanFile, tools
from os import path

class AndroidToolchainConan(ConanFile):
    name = "android-toolchain"
    version = "r16"
from conans import ConanFile, tools
from os import path

class AndroidToolchainConan(ConanFile):
    name = "android-toolchain"
    version = "r16"
    settings = "os_build", "arch_build"
from conans import ConanFile, tools
from os import path

class AndroidToolchainConan(ConanFile):
    name = "android-toolchain"
    version = "r16"
    settings = "os_build", "arch_build"
    buildRequires = "android-ndk/r16@theo/stable"
from conans import ConanFile, tools
from os import path

class AndroidToolchainConan(ConanFile):
    name = "android-toolchain"
    version = "r16"
    settings = "os_build", "arch_build"
    build_requires = "android-ndk/r16@theo/stable"

    def build(self):
        command = "make-standalone-toolchain.sh %s"
        self.run(command % MAKE_TOOLCHAIN_ARGS)
from conans import ConanFile, tools
from os import path

class AndroidToolchainConan(ConanFile):
    def package_info(self):
        sys = path.join(self.package_folder, "sysroot")
        self.cpp_info.sysroot = sys
Android profile

settings
os=Android
arch=armv8
# etc, etc...

build_requires
android-toolchain/r16@theo/stable
Conan & Android

FINAL DEMO