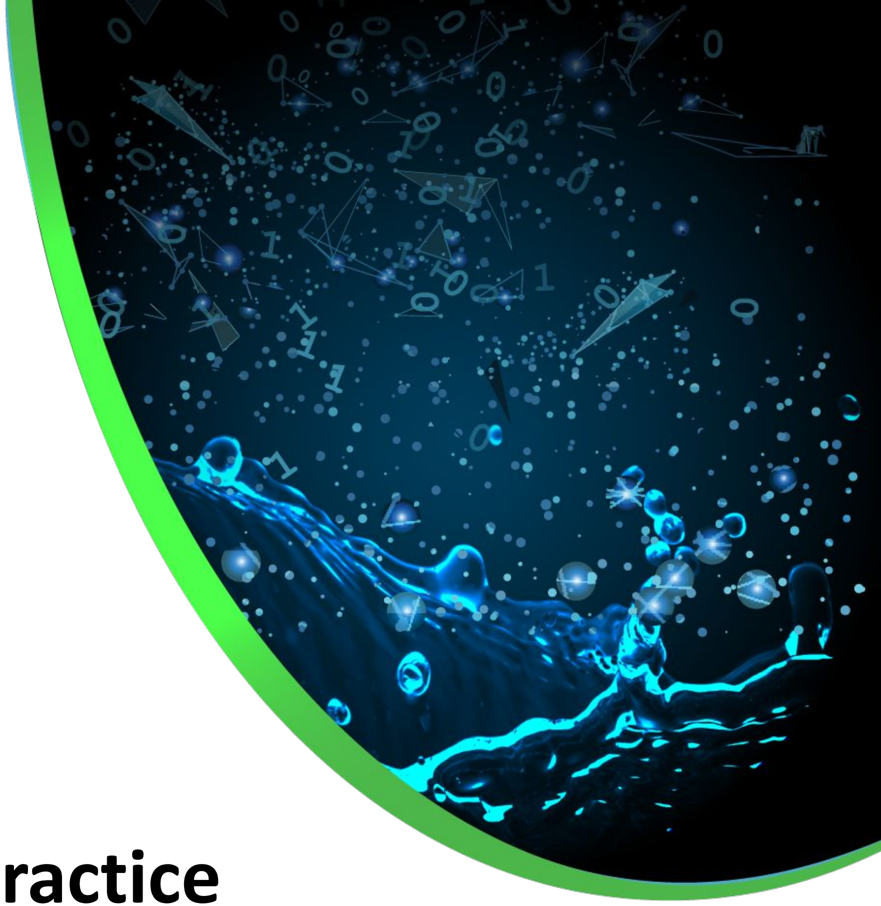
Jerry Wiltse
Software Developer :: Conan Team

CONAN
C/C++ Package manager

Conan Package Manager in Practice



Environment Setup

```
$ git clone https://github.com/solvingj/conan_cpp_demo
$ cd conan_cpp_demo
$ docker-compose up -d
$ docker exec -it conan-terminal-demo bash
# can re-run above command from new shell if disconnected
```

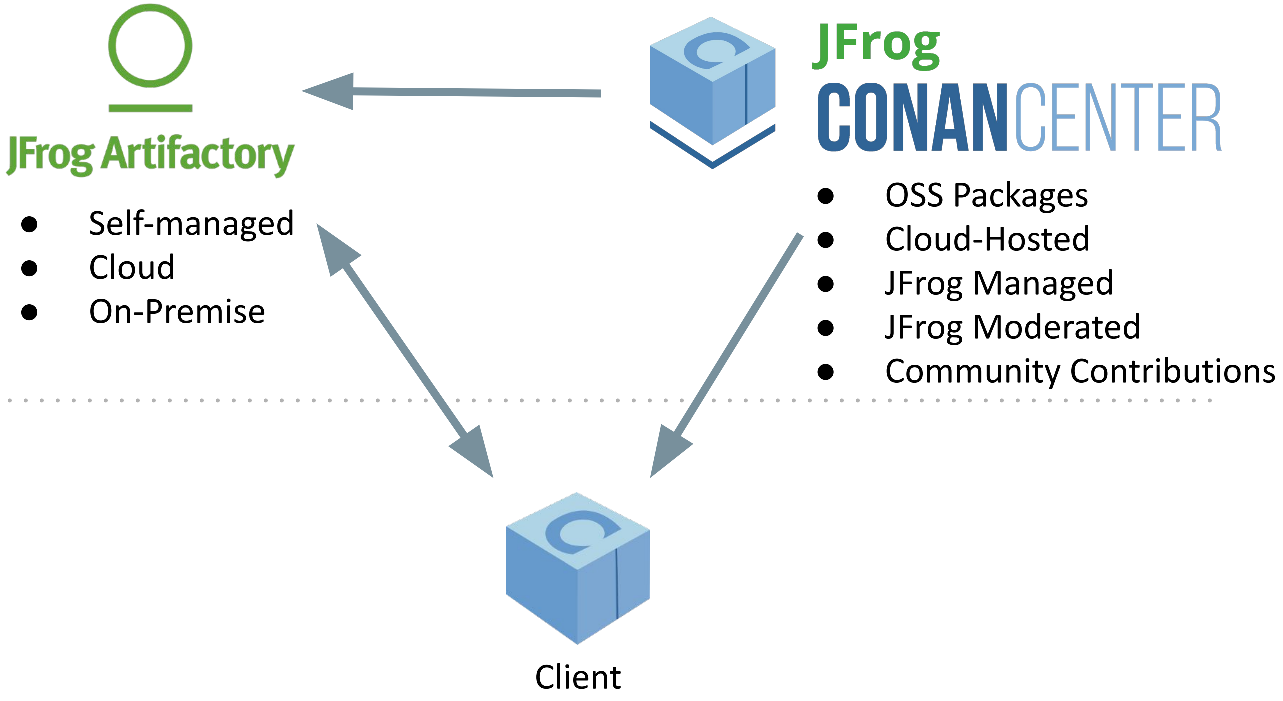
Introduction

- Package Manager for C/C++
- Open-source, MIT license
- Multi-platform
- Any build system
- Stable
- Active
- Free Training Provided by JFrog
  - https://academy.jfrog.com

Architecture

Servers
(artifact storage)

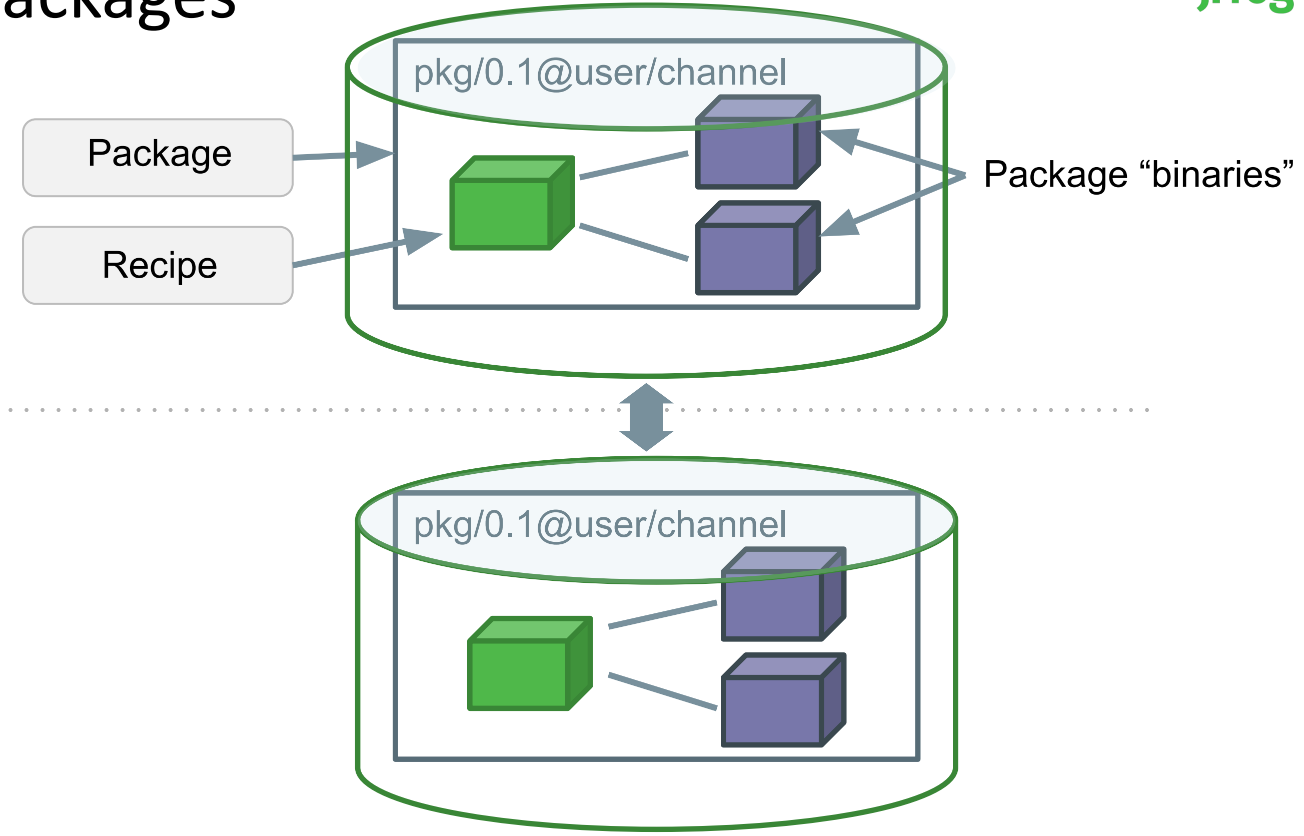
Developer
machine / CI



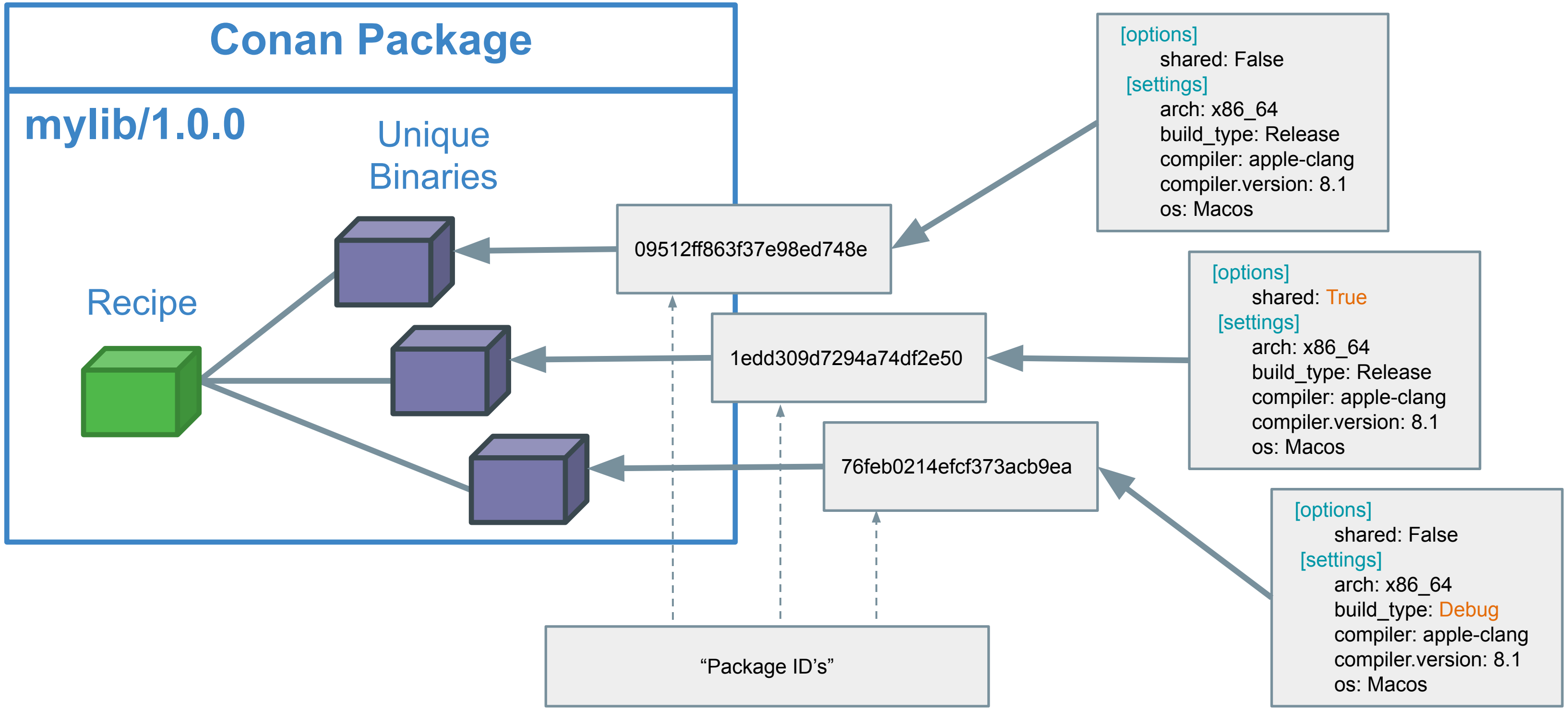
Multi-Binary Packages

Servers

Client



Multi-Binary Packages

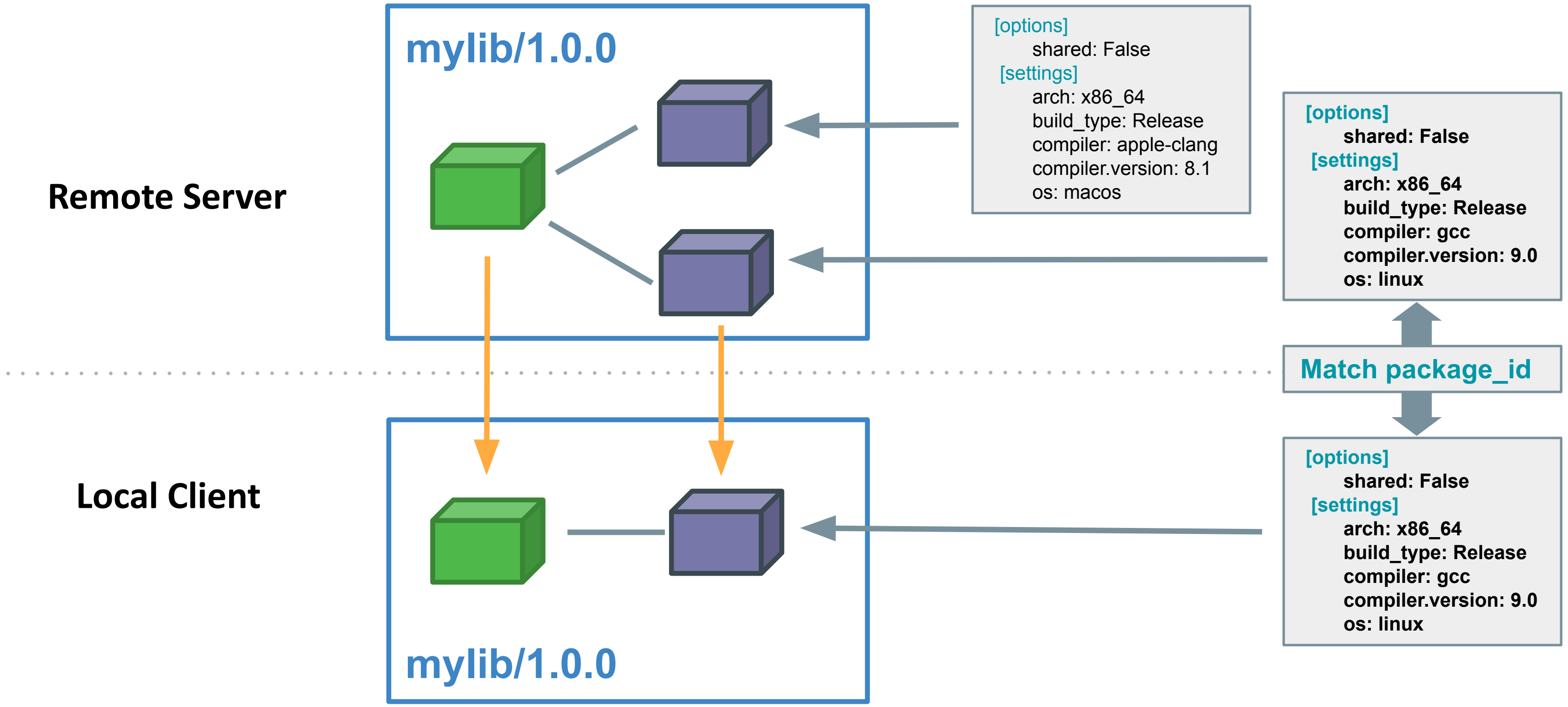


```
[options]
    shared: False
 [settings]
    arch: x86_64
    build_type: Release
    compiler: apple-clang
    compiler.version: 8.1
    os: Macos
```

```
[options]
    shared: True
 [settings]
    arch: x86_64
    build_type: Release
    compiler: apple-clang
    compiler.version: 8.1
    os: Macos
```

```
[options]
    shared: False
 [settings]
    arch: x86_64
    build_type: Debug
    compiler: apple-clang
    compiler.version: 8.1
    os: Macos
```

Remote Repositories

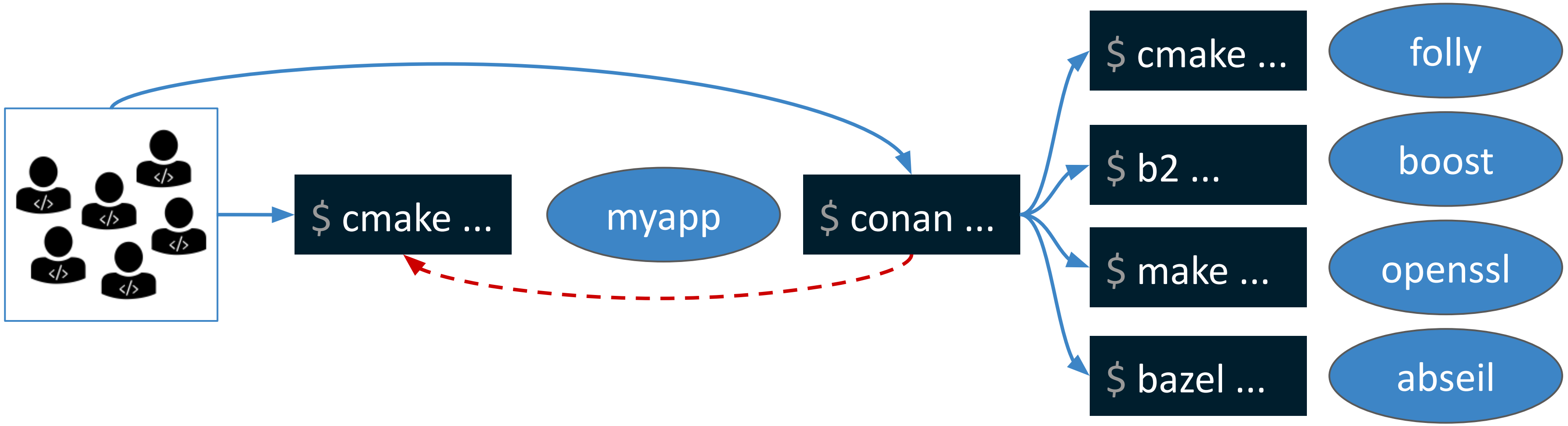


```
[options]
    shared: False
 [settings]
    arch: x86_64
    build_type: Release
    compiler: apple-clang
    compiler.version: 8.1
    os: macos
```

```
[options]
    shared: False
 [settings]
    arch: x86_64
    build_type: Release
    compiler: gcc
    compiler.version: 9.0
    os: linux
```

```
[options]
    shared: False
 [settings]
    arch: x86_64
    build_type: Release
    compiler: gcc
    compiler.version: 9.0
    os: linux
```

Abstracting away build systems for consumers



Exercise : Consume a Conan Package

- Single-file C++ executable
- CMake Build System
- Depends on Boost Regex library
- Command : “conan install ..”

Exercise : Consume a Conan Package

regex.cpp

```
#include <boost/regex.hpp>
#include <string>
#include <iostream>

...
```

conanfile.txt

```
[requires]
boost/1.74.0

[generators]
cmake_find_package
virtualenv
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.1)

project(boost_regex_demo)

find_package(Boost COMPONENTS regex REQUIRED)

add_executable(regex_exe regex.cpp)

target_link_libraries(regex_exe
    PRIVATE
      Boost::regex
)
```

Exercise : Consume a Conan Package : Linux

```
$ cd examples/cmake_find_package
$ mkdir build_linux && cd build_linux
$ conan install .. --profile ../../profiles/linux_gcc_7_release
$ source activate.sh
$ cmake ..  -DCMAKE_BUILD_TYPE=Release  -DCMAKE_MODULE_PATH=$PWD
$ cmake --build .
$ ./regex_exe "Subject: Re: conan"
> Regarding : conan
$ source deactivate.sh
$ cd ..
# Above uses pre-compiled binaries from conan-center
# Alternatively, build some, or all dependencies from source
$ conan install .. --build=all          # or --build=boost,bzip2
```

Exercise : Consume a Conan Package : Windows

```
$ cd examples/cmake_find_package
$ mkdir build_windows && cd build_windows
$ conan install .. --profile ../../profiles/windows_msvc_16_release
$ call activate.bat
$ cmake ..  -DCMAKE_BUILD_TYPE=Release  -DCMAKE_MODULE_PATH=%CD:\=/%
$ cmake --build . --config Release
$ Release\regex_exe.exe "Subject: Re: conan"
> Regarding : conan
$ call deactivate.bat
$ cd ..
# Above uses pre-compiled binaries from conan-center
# Alternatively, build some, or all dependencies from source
$ conan install .. --build=all          # or --build=boost,bzip2
```

Exercise : Consume a Conan Package : Summary

- Command: “conan install”
- Consuming OSS packages can be simple
- Can provide dependencies to any build system
  - Including support cmake find_package”
    - Including Components Support for Boost, etc.
- Conan Center provides many OSS packages
  - Many precompiled binaries
  - --build=... to build dependencies from source
    - Often recommended or required

Consuming OSS libraries is only half of C/C++

- Other half is private dependency management
  - Professional development teams
    - Enterprise, Startup, Research, Academia
  - At least as many affected users as the OSS community
  - At least as complicated as OSS development
  - Completely new sets of challenges
    - Scalability, Maintainability, Reproducibility, Etc.
- Conan has extensive collection of related features
  - Devoting at least as much time to these use-cases

What is a Conan Recipe?

- Recipe is the instruction file to create a package
  - “conanfile.py” (a python class)
- Show Three Examples
  - Empty Recipe
  - Example with CMake project
  - Example for generic/custom build system

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

class MylibConan(ConanFile):
    name = "mylib"
    version = "0.1.0"

    def requirements(self):
        # define dependencies

    def export_sources(self):
        # capture the sources

    def generate(self):
        # convert conan variables into build-system files

    def build(self):
        # invoke the build system, reading generated files

    def package(self):
      # copy artifacts from “build” to “package” directory

    def package_info(self):
      # declare whats in the package for consumers
```

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

class MylibConan(ConanFile):
    name = "mylib"
    version = "0.1.0"
    settings = "os", "arch", "compiler", "build_type"

    def requirements(self):
        self.requires("boost/1.74.0@")    # -> depend on boost 1.74.0

    def export_sources(self):
        self.copy("*")                     # -> copies all files/folders from working dir into a “source” directory

    def generate(self):
        CMakeToolchain(self).generate()    # -> conantoolchain.cmake  (variables translated from conan settings)
        CMakeDeps(self).generate()         # -> creates FindBoost.cmake  (sets paths to Boost files in conan cache)

    def build(self):
        cmake = CMake(self)                # CMake helper auto-formats CLI arguments for CMake
        cmake.configure()                  # cmake -DCMAKE_TOOLCHAIN_FILE=conantoolchain.cmake
        cmake.build()                      # cmake --build .

    def package(self):
        cmake = CMake(self)                # For CMake projects which define an install target, leverage it
        cmake.install()                    # cmake --build . --target=install
                                           # sets CMAKE_INSTALL_PREFIX to appropriate directory in conan cache

    def package_info(self):
        self.cpp_info.includedirs = ["include"]     # List of header directories
        self.cpp_info.libdirs = ["lib"]             # List of directories to search for libraries
        self.cpp_info.libs = ["mylib"]              # List of libraries to link with
```

```python
from conans import ConanFile

class MyLibConan(ConanFile):
    name = "mylib"
    version = "0.1.0"
    settings = "os", "arch", "compiler", "build_type"

    def requirements(self):
        self.requires("boost/1.74.0")    # -> depend on boost 1.74.0

    def export_sources(self):
        self.copy("*")                       # -> copies all files/folders from working dir into a “source” directory

    def generate(self):
        self._custom_function()                          # -> customfile.txt (custom code to generate custom file)

    def build(self):
        self.run("custom_build_system … <flags>")         # -> build system read dependency info from customfile.txt

    def package(self):
        self.copy("*.h", dst="include", src="src")        # Organize lib files for each os into uniform dir structure
        self.copy("*.dll", dst="bin", keep_path=False)
        self.copy("*.lib", dst="lib", 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.includedirs = ["include"]      # List of header directories
        self.cpp_info.libdirs = ["lib"]              # List of directories to search for libraries
        self.cpp_info.libs = ["mylib"]               # List of libraries to link with
```

Exercise : Create a Conan Package

- Same Project as Previous Example
- Replace “conanfile.txt” with “conanfile.py”
  - Define all required methods in conanfile.py
- Create a package from the recipe
  - Command: “conan create”

Exercise : Create a Conan Package

regex.cpp

```
#include <boost/regex.hpp>
#include <string>
#include <iostream>

...
```

conanfile.py

```
Next Slide
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.1)

project(boost_regex_demo)

find_package(Boost COMPONENTS regex REQUIRED)

add_executable(regex_exe regex.cpp)

target_link_libraries(regex_exe
    PRIVATE
      Boost::regex
)
```

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

class RegexConan(ConanFile):
    name = "regex"
    version = "0.1.0"
    settings = "os", "arch", "compiler", "build_type"
    generators = "cmake_find_package", "virtualenv"

    def requirements(self):
        self.requires("boost/1.74.0@")

    def export_sources(self):
        self.copy("*")

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

    def package(self):
        cmake = CMake(self)
        cmake.install()
```

Exercise : Create a Conan Package : Linux

```
$ cd examples/create_package
$ conan create . demo/demo --profile ../profiles/linux_gcc_7_release
$ mkdir run_linux && cd run_linux
$ conan install regex/0.1.0@demo/demo -g virtualrunenv \
     --profile ../profiles/linux_gcc_7_release
$ source activate_run.sh
$ regex_exe "Subject: Re: conan"
> Regarding : conan
$ source deactivate_run.sh
$ cd ..
```

Exercise : Create a Conan Package : Windows

```
$ cd examples/create_package
$ conan create . demo/demo --profile ../profiles/windows_msvc_16_release
$ mkdir run_windows && cd run_windows
$ conan install regex/0.1.0@demo/demo -g virtualrunenv ^
    --profile ../profiles/windows_msvc_16_release
$ activate_run.bat   REM no “source” command on windows
$ regex_exe.exe "Subject: Re: conan"
> Regarding : conan
$ deactivate_run.bat  REM no “source” command on windows
$ cd ..
```

Exercise : Create a Conan Package : Summary

- Conan Recipe : “conanfile.py”
  - Instructions for Creating a Conan Package
- Python Class with Standard Methods
  - requirements()
  - exports_sources()
  - build()
  - package()
  - package_info()
- Conan calls methods in order to create a package

Exercise : Upload a Conan Package

- “conan remote list” shows all remotes
- “conan remote add” to add new remotes
- add repository from demo docker environment
  - JFrog Artifactory CE for C/C++
    - Free Community Edition
    - Designed for Conan Repositories
- Command: “conan upload”

Environment Setup : Start Artifactory CE

```
$ docker-compose -f docker-compose-artifactory-ce.yml up -d
$ docker exec -it conan-terminal-demo bash
# can re-run above command from new shell if disconnected
```

Exercise : Upload a Conan Package

```
$ conan remote list
$ conan remote add artifactory  \
        http://artifactory-ce-demo:8081/artifactory/api/conan/conan-local

$ conan user -p=password -r=artifactory admin   # Login to Remote

$ conan upload "regex/0.1.0@demo/demo" -r=artifactory --all
$ conan search regex/0.1.0@demo/demo -r=artifactory
```

Exercise : Upload a Conan Package : Summary

- Command: “conan upload”
- Other easy commands for remote management
- conan-center comes installed by default
- Local conan cache in ~/.conan/data
    - Shared by any number of local projects and builds
- Local/Remote repository strategy similar to other package managers
- Artifactory CE for C/C++
    - Free, local hosting for Conan repositories

More Resources

1 Github Project

2 Blog

3 Documentation

4 Conan-Center

5 JFrog Academy Courses

6 Slack: https://cppalliance.org/slack #conan

THANK YOU!

JFrog | CONAN
C/C++ Package manager

@conan_io @solvingj linkedin.com/company/conan-io