Welcome!

PREREQUISITES

- Wi-Fi enabled Mac or PC
- SSH client
- Internet Browser

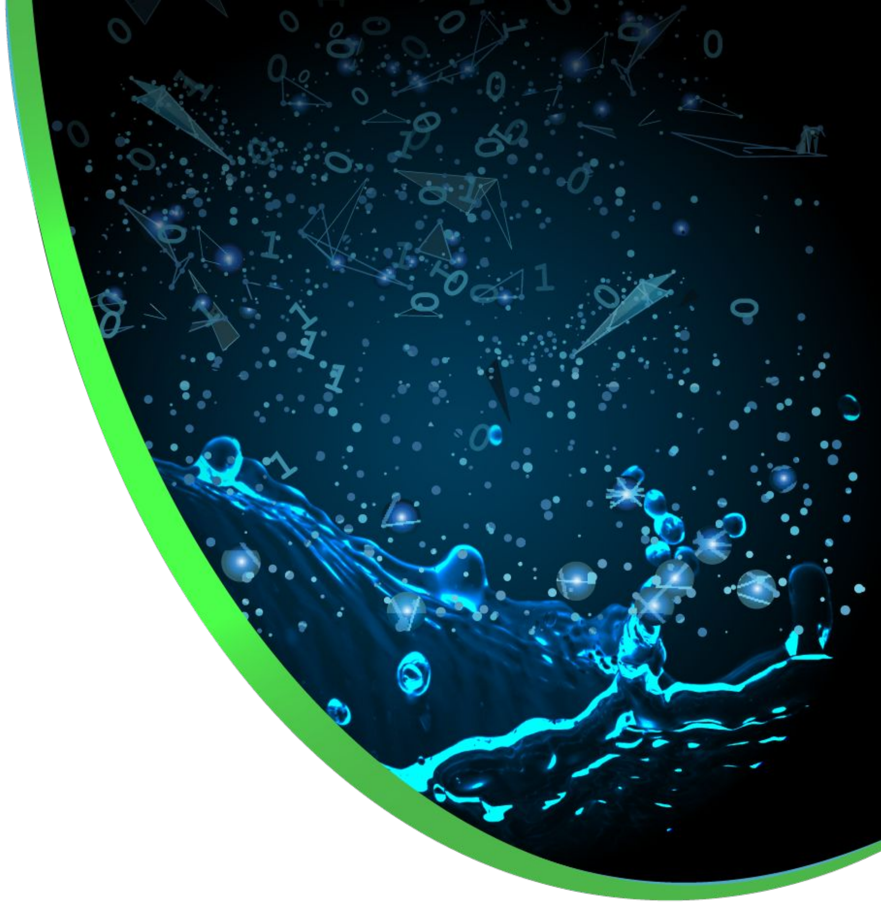
1. Register and launch https://jfrog.orbitera.com/c2m/trial/1289
2. ssh conan@IP

cppcon
the c++ conference
SEPTEMBER 15-20 2019
Aurora, Colorado, USA

CONAN
C/C++ Package manager

Introduction to Conan C++ Package Manager

Diego Rodriguez-Losada, Conan Founder
Luis Martinez de Bartolome, Conan Founder



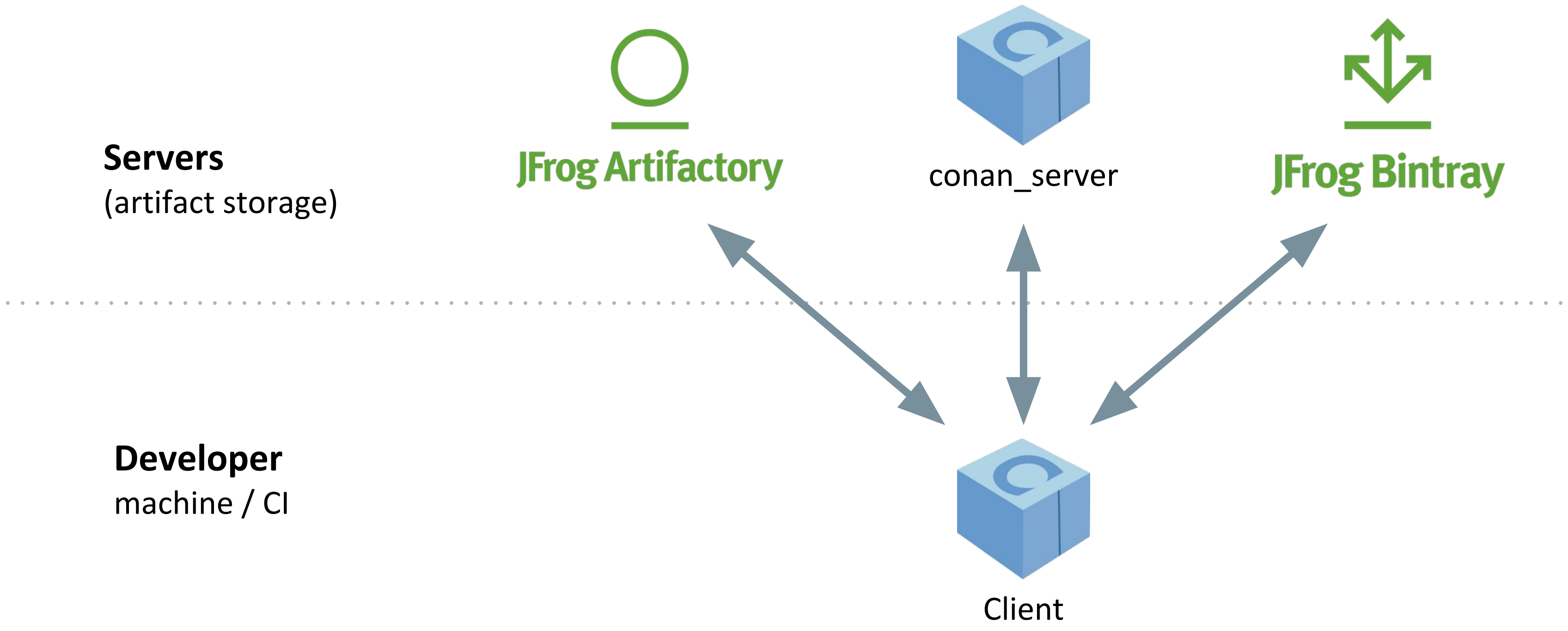
Outline

• **Introduction** • Consume Conan packages • Create Conan packages • Uploading packages to Artifactory	Part I
• Build configuration & cross-build • Requirements • Hooks and Conan configuration • Versioning	Part II
• Jenkins Artifactory Conan CI	Bonus

Introduction

- OSS, MIT license
- Multi-platform
- Any build system
- Stable
- Active

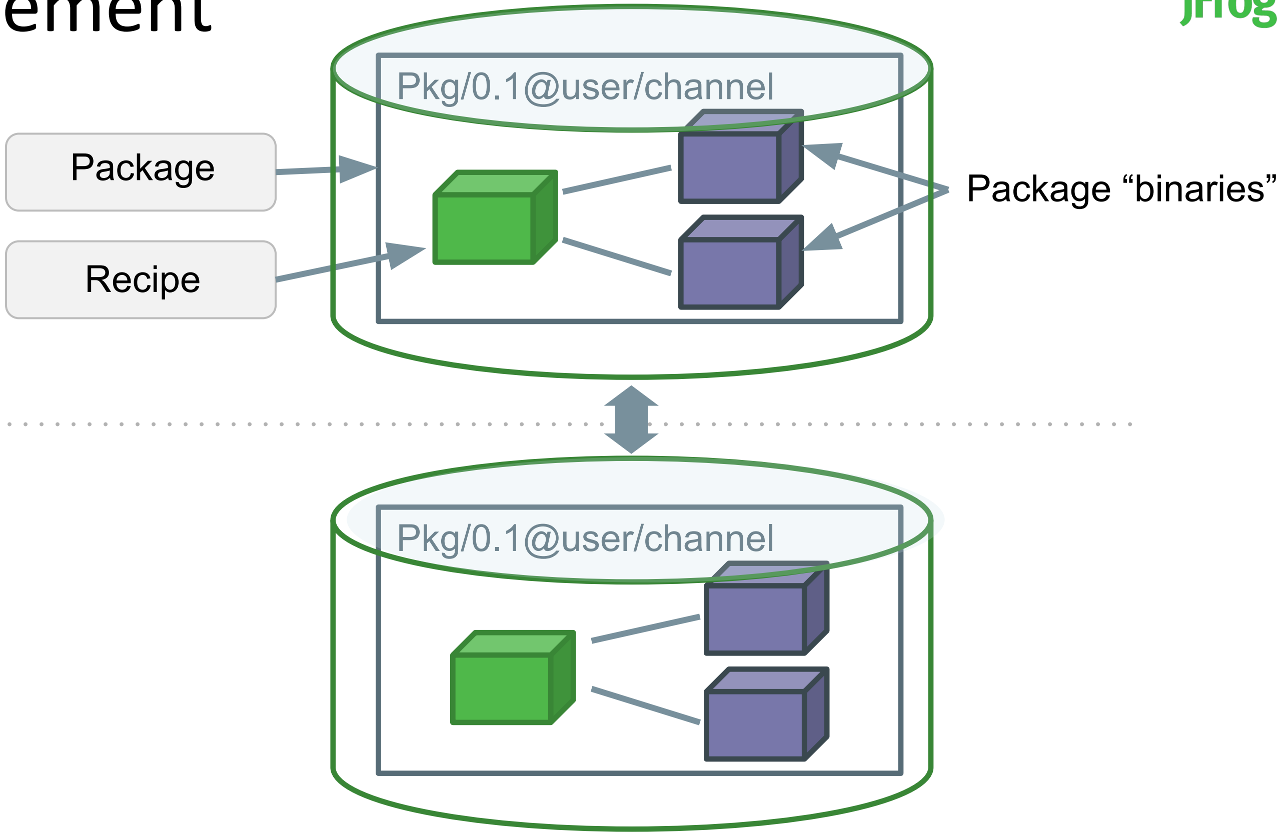
Architecture



Binary Management

Servers

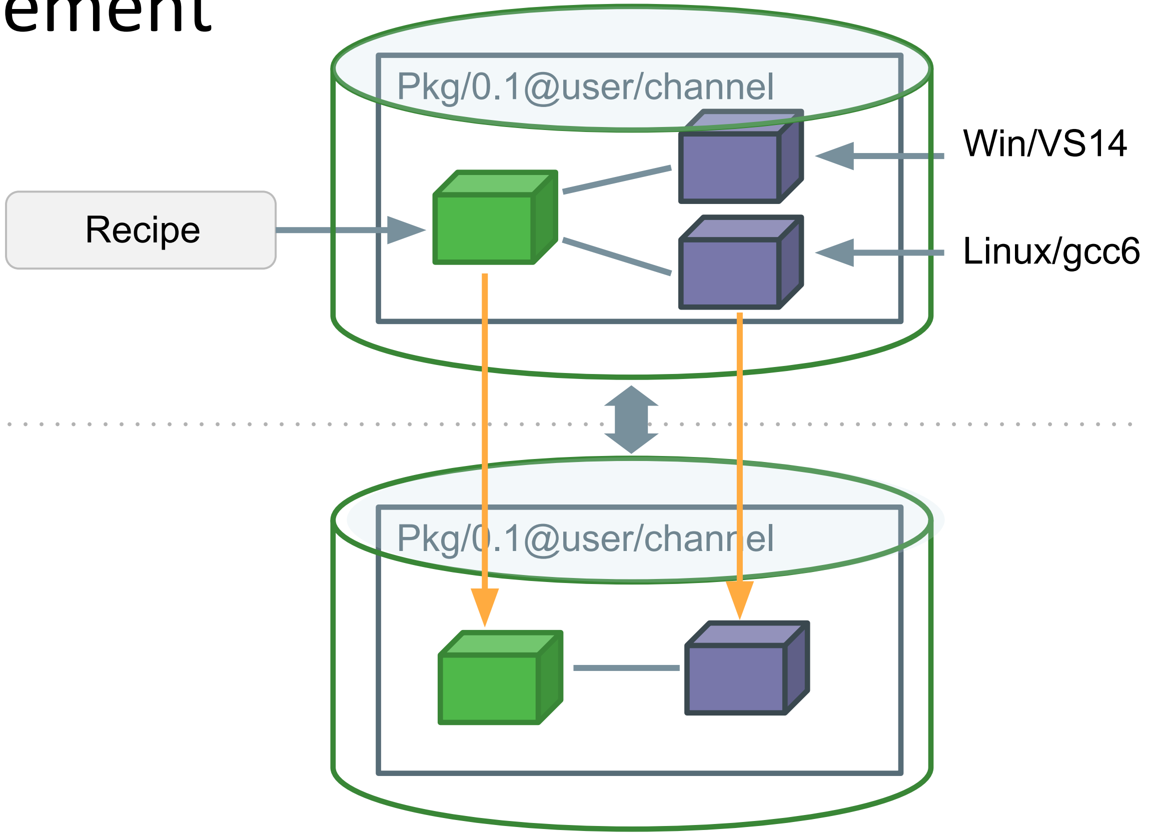
Client



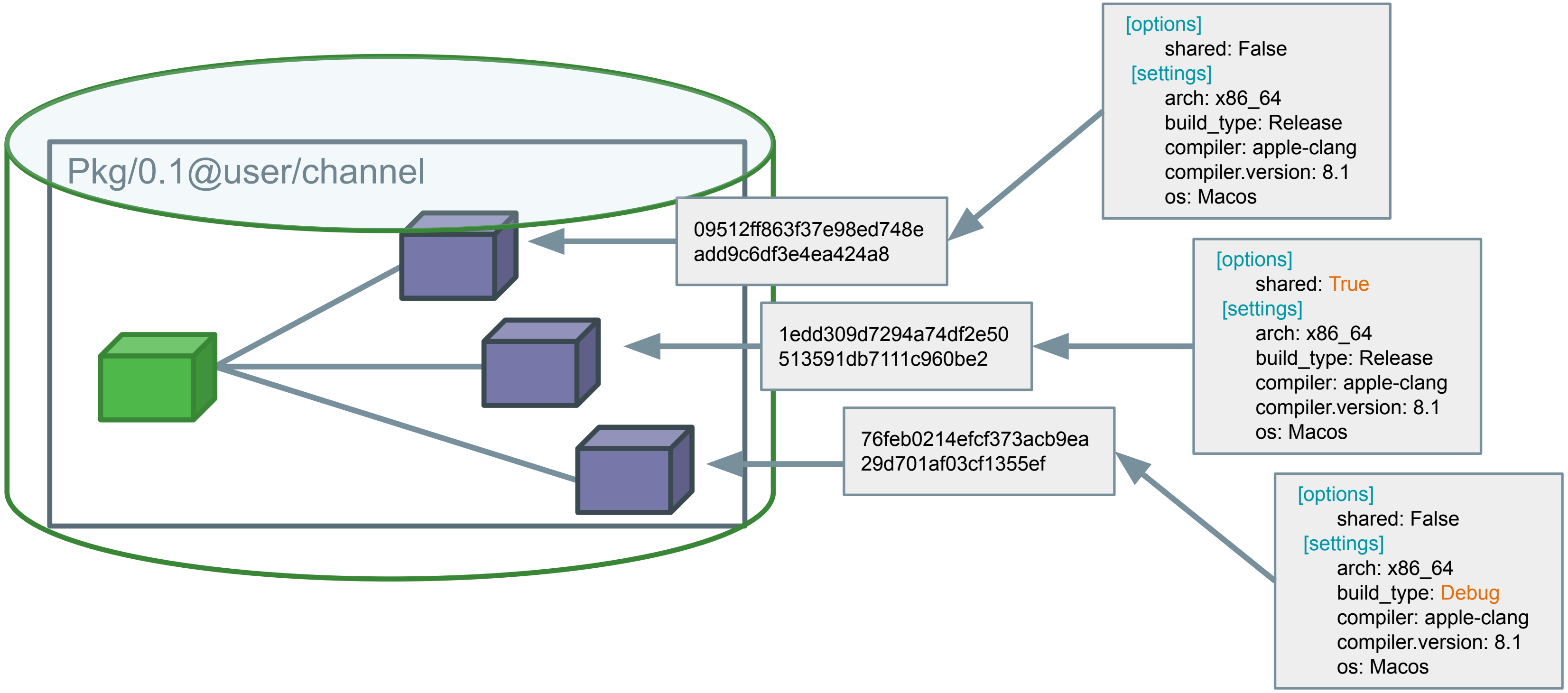
Binary Management

Servers

Client



Binary Management



```
[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
```

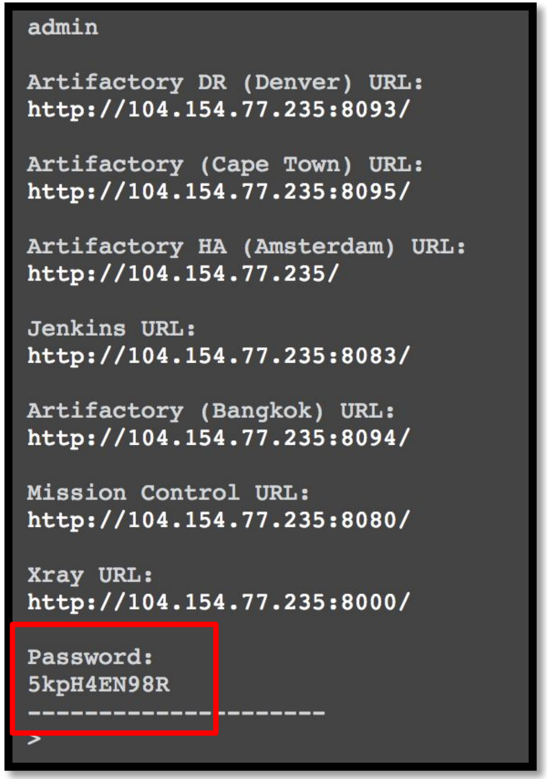
```
[options]
    shared: False
[settings]
    arch: x86_64
    build_type: Debug
    compiler: apple-clang
    compiler.version: 8.1
    os: Macos
```

Outline

- Introduction
- **Consume Conan packages**
- Create Conan packages
- Uploading packages to Artifactory
- Build configuration & cross-build
- Requirements
- Hooks and Conan configuration
- Versioning
- Jenkins Artifactory Conan CI

Exercise 1 - Setup

```
# https://jfrog.orbitera.com/c2m/trial/1289

 $ ssh conan@<orbitera-IP>
# Use password from orbitera
$ git clone https://github.com/conan-io/training
```

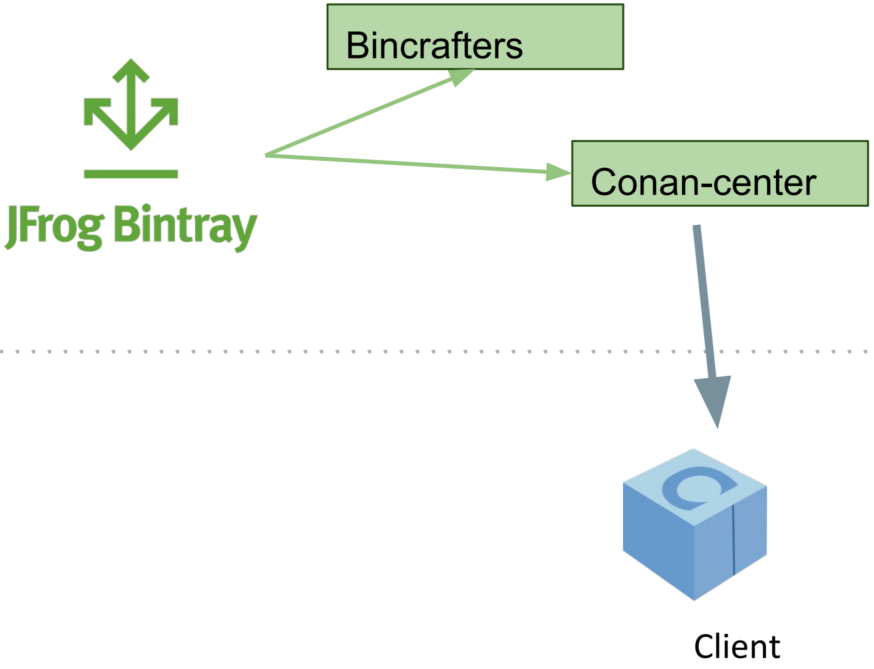


```
admin

Artifactory DR (Denver) URL:
http://104.154.77.235:8093/

Artifactory (Cape Town) URL:
http://104.154.77.235:8095/

Artifactory HA (Amsterdam) URL:
http://104.154.77.235/

Jenkins URL:
http://104.154.77.235:8083/

Artifactory (Bangkok) URL:
http://104.154.77.235:8094/

Mission Control URL:
http://104.154.77.235:8080/

Xray URL:
http://104.154.77.235:8000/

Password:
5kpH4EN98R
---------------------
>
```

Exercise 2 - Consume

Servers
(artifact storage)

Developer
machine / CI

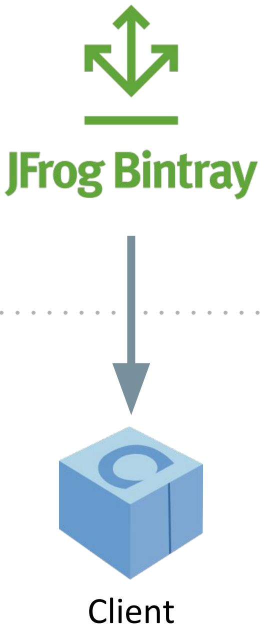


Exercise 2 - Consume

Servers
(artifact storage)

https://bintray.com/conan/conan-center

Developer
machine / CI



Exercise 2 – Consume with CMake

```
$ cd training/consumer
$ vim/nano timer.cpp
```

timer.cpp

```cpp
#include "Poco/Timer.h"
#include "Poco/Thread.h"
#include "Poco/Stopwatch.h"

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

...
```

conanfile.txt

```
[requires]
boost/1.67.0@conan/stable
Poco/1.9.0@pocoproject/stable

[generators]
cmake

[options]
Boost:shared=False
Poco:shared=False
```

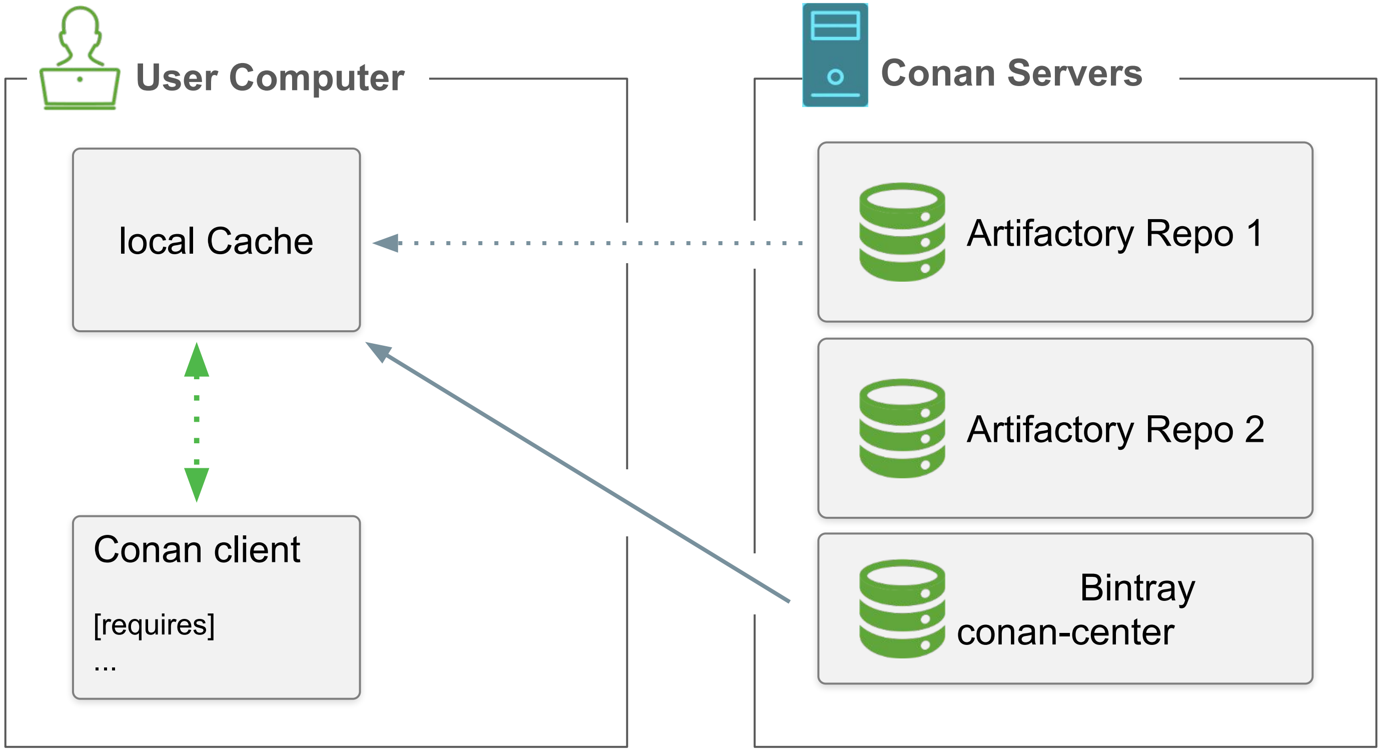
CMakeLists.txt

```cmake
cmake_minimum_required(VERSION 2.8.12)
project(BoostPoco)
add_compile_options(-std=c++11)

# Using the "cmake" generator
include(${CMAKE_BINARY_DIR}/
            conanbuildinfo.cmake)
conan_basic_setup()

add_executable(timer timer.cpp)
target_link_libraries(timer ${CONAN_LIBS})
```

Exercise 2 - Consume with CMake

```
$ mkdir build && cd build
$ conan install ..
# check the generated conanbuildinfo.cmake
$ vim conanbuildinfo.cmake
$ cmake .. –DCMAKE_BUILD_TYPE=Release
$ cmake --build .    # or make
$ bin/timer
>…
                                        $ ../catchup.sh # option 2
```

Exercise 2 – How Conan Installs Packages



Installed Packages (search)

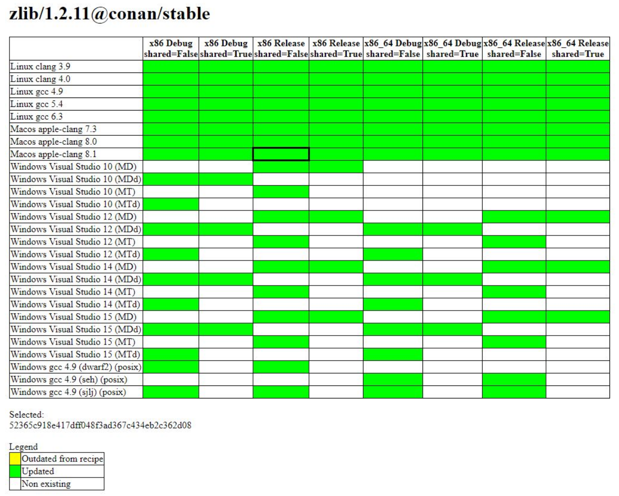
```
$ conan search
$ conan search zlib/1.2.11@conan/stable
```

Exercise 3 - Consume (debug mode)

```
$ conan install .. – s build_type=Debug
# note that new packages are installed
$ cmake .. –DCMAKE_BUILD_TYPE=Debug
$ cmake --build .
$ bin/timer
>…
$ conan search zlib/1.2.11@conan/stable

                                                  $ ../catchup.sh # option 3
```

Conan Info & Search

```
$ conan search
$ conan search zlib/1.2.11@conan/stable   # add --table=file.html
$ conan info ..   # --graph=file.html
```



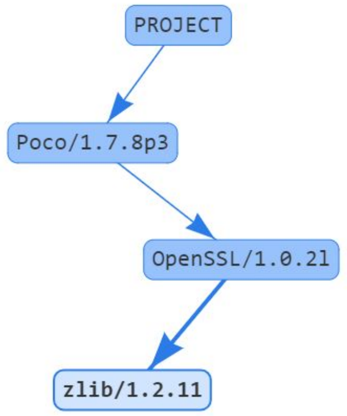
zlib/1.2.11@conan/stable

	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



$$\begin{tikzcd}
\text{PROJECT} \arrow[d] \\
\text{Poco/1.7.8p3} \arrow[d] \\
\text{OpenSSL/1.0.2l} \arrow[d] \\
\text{zlib/1.2.11}
\end{tikzcd}$$

Exercise 4 - Consume (gcc generator)

```
$ cd training/consumer_gcc
$ ls # Look Ma, no build system!
$ conan install . – g gcc # -g=compiler_args too
# check conanbuildinfo.gcc
$ g++ timer.cpp @conanbuildinfo.gcc -o timer -std=c++11
$ ./timer
>...
                                        $ ../catchup.sh # option 4
```

Generators

- Visual Studio
    - Legacy
    - Multi
- Cmake
    - Multi
    - cmake_paths, cmake_find_package, cmake_find_package_multi
- XCode
- pkg-config
- boost
- qmake, qbs, premake
- virtualrunenv, virtualbuildenv
- YOUR OWN!

Exercise 5 – Consume with modern CMake

```
$ cd training/consumer
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8)
project(BoostPoco)
add_compile_options(-std=c++11)

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

add_executable(timer timer.cpp)
target_link_libraries(timer  CONAN_PKG::Poco
                             CONAN_PKG::boost)
```

Exercise 5 - Consume with modern CMake

```
$ cd build
$ cmake --build .    # or make
$ ./timer  # was bin/timer, but no more bc NO_OUTPUT_DIRS
>…
```

Exercise 6 – Consume with CMake find

```
$ cd consumer_cmake_find
```

conanfile.txt

```
[requires]
boost/1.67.0@conan/stable
Poco/1.9.0@pocoproject/stable

[generators]
cmake_find_package

[options]
Boost:shared=False
Poco:shared=False
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.0)
project(BoostPoco)
add_compile_options(-std=c++11)

# Using the "cmake_find_package" generator
set(CMAKE_MODULE_PATH ${CMAKE_BINARY_DIR} ${CMAKE_MODULE_PATH})
set(CMAKE_PREFIX_PATH ${CMAKE_BINARY_DIR} ${CMAKE_PREFIX_PATH})

find_package(boost REQUIRED)
find_package(Poco REQUIRED)

add_executable(timer timer.cpp)
target_link_libraries(timer Poco::Poco boost::boost)
```

Exercise 6 - Consume with CMake find

```
$ mkdir build && cd build
$ conan install .. # check the generated Findxxxx.cmake
$ cmake .. –DCMAKE_BUILD_TYPE=Release
$ cmake --build .    # or make
$ timer
>...
```

Outline

- Introduction
- Consume Conan packages
- **Create Conan packages**
- Uploading packages to Artifactory
- Build configuration & cross-build
- Requirements
- Hooks and Conan configuration
- Versioning
- Jenkins Artifactory Conan CI

Exercise 7 – Create Package (from github src)

- “Hello” library in https://github.com/conan-io/hello.git
- All we need is a “recipe”:
    - source
    - build
    - package
    - package info

```python
class HelloConan(ConanFile):
    name = "hello"
    version = "0.1"
    settings = "os", "compiler", "build_type", "arch"
    generators = "cmake"

    def source(self):
        self.run("git clone https://github.com/conan-io/hello.git")

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

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

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

Exercise 7 – Create Package (from github src)

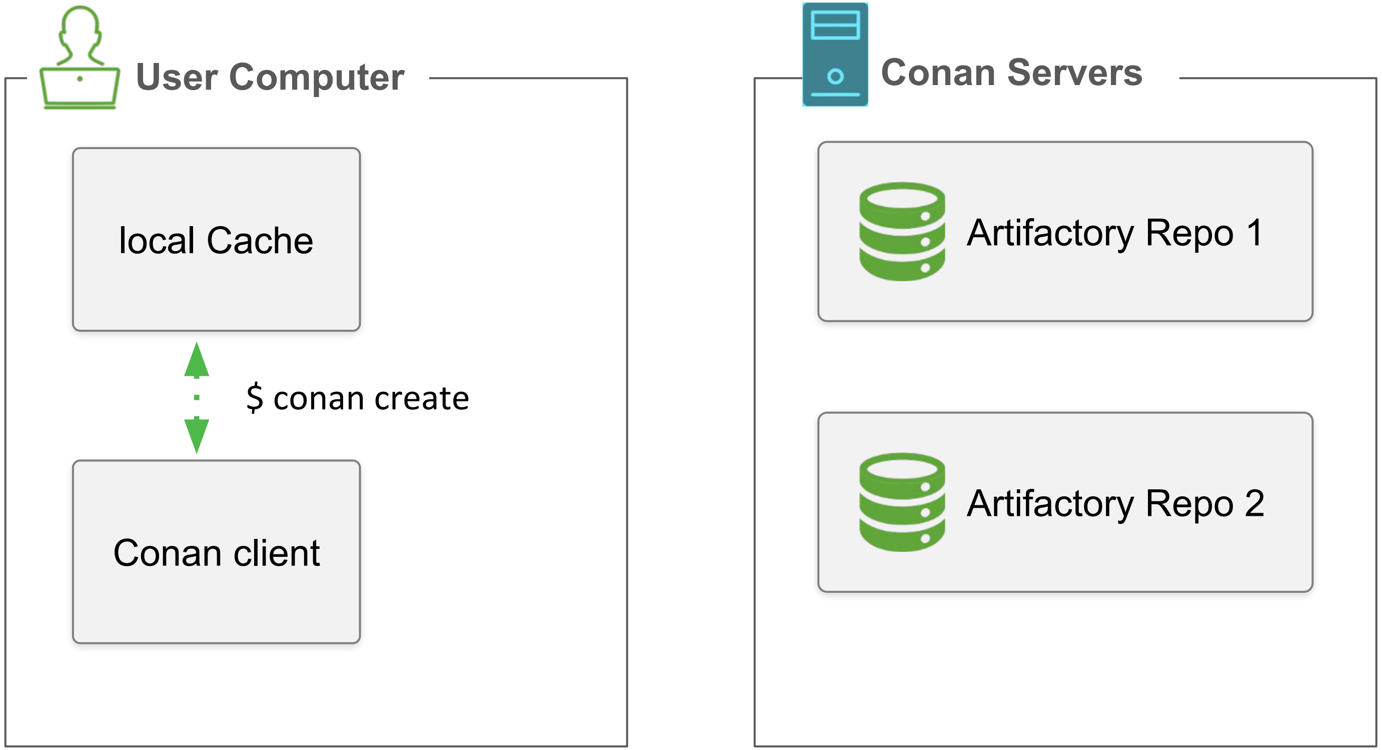
```
$ cd ../create
$ conan new hello/0.1  # just a template
# check the conanfile.py
$ conan create . user/testing
> …
$ conan search
$ conan search hello/0.1@user/testing
```

Fetching the sources from: https://github.com/conan-io/hello

Exercise 7 – Create Package (from github src)

```
$ conan create . user/testing –s build_type=Debug
> …
$ conan search hello/0.1@user/testing
```

Conan Create is Local



Exercise 8 – Consume “hello” package

```
$ cd consumer
# modify code to include and call the hello() function
# modify conanfile.txt to account for new dependency
# conan install to update dependency graph and conanbuildinfo.cmake
# build and run again
```

Exercise 9 – Create & test package

```
$ cd training/create
$ conan new hello/0.1 -t # The –t generates test_package
```

test_package/conanfile.py (consumer)

```
class HelloTestConan(ConanFile):
    settings = "os", "compiler", "build_type", "arch"
    generators = "cmake"
    # No require necessary

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


    def test(self):
        if not tools.cross_building(self.settings):
            os.chdir("bin")
            self.run(".%sexample" % os.sep)
```

test_package/example.cpp

```
#include <iostream>
#include "hello.h"

int main() {
    hello();
}
```

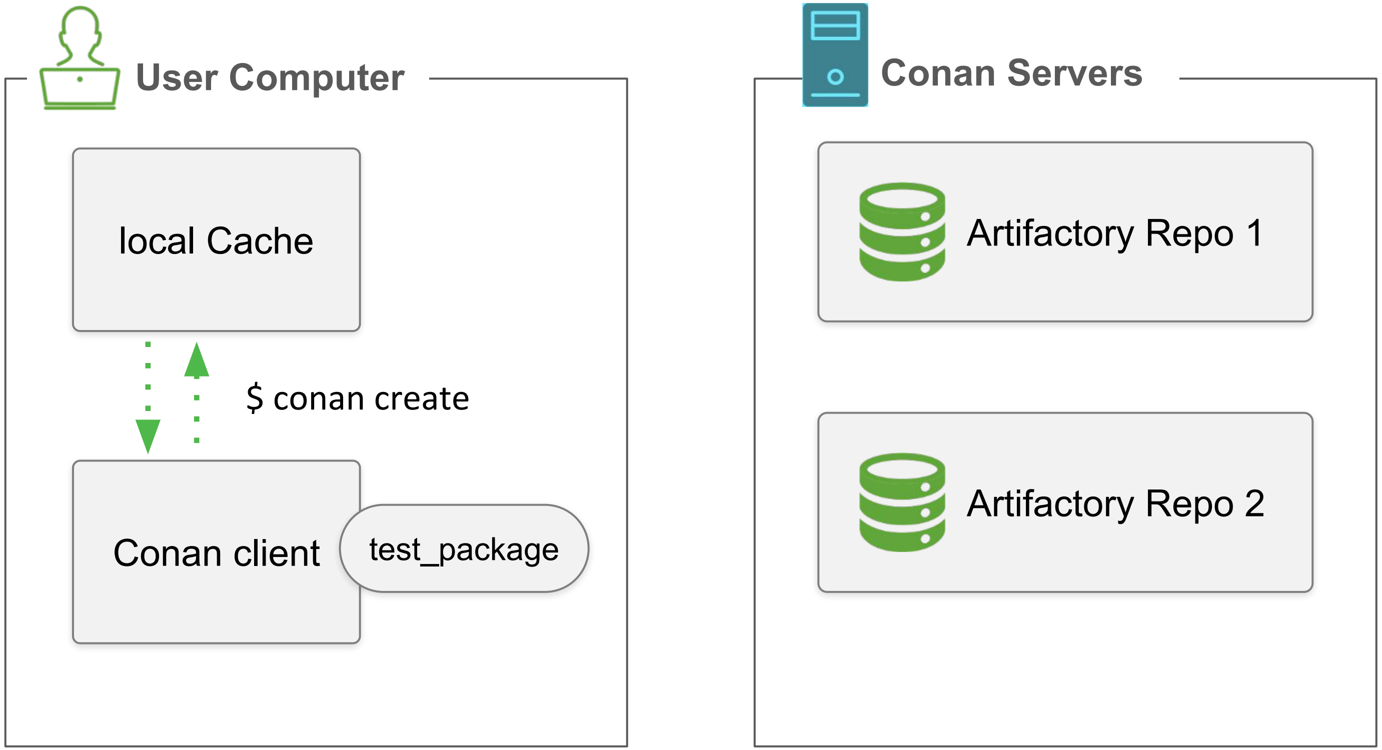
Exercise 9 – Create & test package

```
$ conan new hello/0.1 -t # -t generates test_package
$ conan create . user/testing
> …# check output
> Hello World!
```

Exercise 9 – Create & test package

```
$ conan create . user/testing  -s build_type=Debug
> …# check output
> Hello World!

                                   $ ../catchup.sh # option 9
```

Conan Create (with test_package) is Local



Exercise 10 – Create (from src repo)

```
$ cd training/create_sources
$ conan new hello/0.1 -t -s # The –s generates example src
```

conanfile.py

```
class HelloConan(ConanFile):
    name = “hello"
    version = “0.1"
    def build(self):
    def package(self):
    def package_info(self):
```

src/CMakeLists.txt

```
project(MyHello CXX)
cmake_minimum_required(VERSION 2.8)

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

add_library(hello hello.cpp)
```

src/hello.h & src/hello.cpp

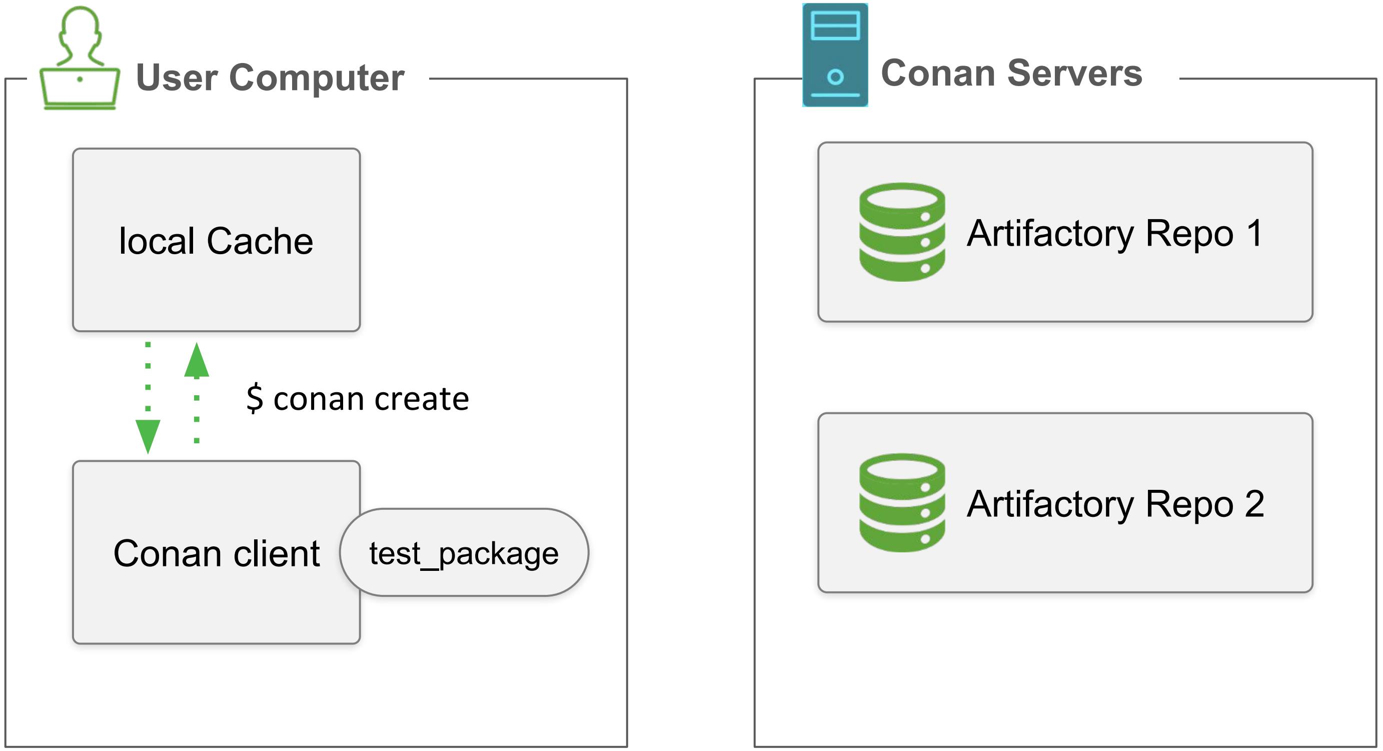
```
#include <iostream>
#include "hello.h"

void hello(){
 #ifdef NDEBUG
    std::cout << "Hello World Release!"
<<std::endl;
#else
    std::cout << "Hello World Debug!"
<<std::endl;
 #endif
}
```

```
class HelloConan(ConanFile):
    name = "hello"
    version = "0.1"
    settings = "os", "compiler", "build_type", "arch"
    generators = "cmake"
    exports_sources = "src/*"

    # NO SOURCE METHOD

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

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

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

Exercise 10 – Create (from src repo)

```
$ conan create . user/testing
> …# check output
> Hello World Release!
$ conan create . user/testing –s build_type=Debug
> Hello World Debug!

$ ../catchup.sh # option 10
```

Conan Create (with test_package) is Local



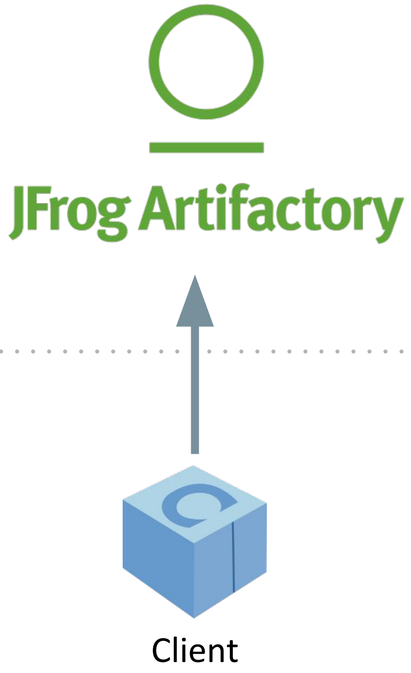
Outline

- Introduction
- Consume Conan packages
- Create Conan packages
- **Uploading packages to Artifactory**
- Build configuration & cross-build
- Requirements
- Hooks and Conan configuration
- Versioning
- Jenkins Artifactory Conan CI

Exercise 11 – Upload to Artifactory

Servers
(artifact storage)

Developer
machine / CI

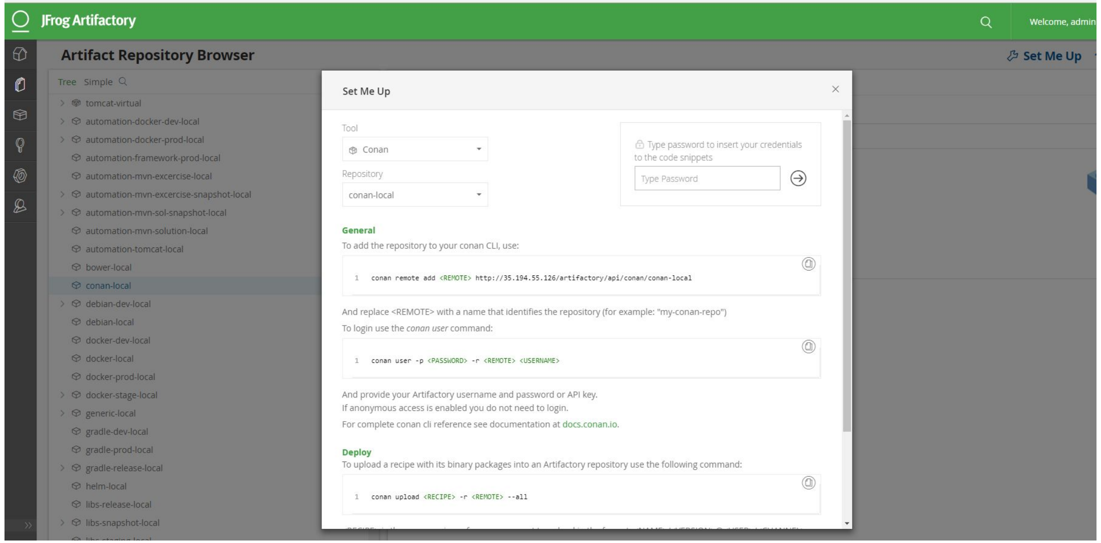


Conan Remotes

```
$ conan remote list
```

Artifactory

- Navigate to IP
  - Admin->Repositories->Local->New
- Create new conan repo **"myconanrepo"**
- Navigate to "Artifact browser"
  - Set Me Up



Set Me Up

Tool: Conan

Repository: conan-local

Type password to insert your credentials to the code snippets

Type Password

General

To add the repository to your conan CLI, use:

```
conan remote add <REMOTE> http://35.194.55.126/artifactory/api/conan/conan-local
```

And replace <REMOTE> with a name that identifies the repository (for example: "my-conan-repo")

To login use the *conan user* command:

```
conan user -p <PASSWORD> -r <REMOTE> <USERNAME>
```

And provide your Artifactory username and password or API key.
If anonymous access is enabled you do not need to login.

For complete conan cli reference see documentation at docs.conan.io.

Deploy

To upload a recipe with its binary packages into an Artifactory repository use the following command:

```
conan upload <RECIPE> -r <REMOTE> --all
```

Exercise 11 – Upload Packages to Artifactory

```
$ conan remote add artifactory <URL from SetMeUp>
$ conan upload "hello*" -r artifactory --all
$ conan search "*" -r=artifactory
$ conan search hello/0.1@user/testing -r=artifactory
# Navigate to Artifactory WebUI and check!
```

Exercise 11 – Upload ALL Packages to Artifactory

```
$ conan upload "*" -r artifactory --all --confirm
$ conan search "*" -r=artifactory
# Navigate to Artifactory WebUI and check!

# We could: $ conan remote remove conan-center
$ conan remove "*" –f
```

```
$ ../catchup.sh # option 11
```

Exercise 12 – Consume packages from Artifactory

```
$ cd consumer
$ mkdir build && cd build
$ conan install .. -r=artifactory
$ cmake .. –DCMAKE_BUILD_TYPE=Release
$ cmake --build .    # or make
$ bin/timer
>…

                                   $ ../catchup.sh # option 12
```

Exercise 13 – Test Uploaded Packages

```
$ cd create_sources
$ conan remove "hello*" -f
$ conan test test_package hello/0.1@user/testing
> …
$ conan test test_package hello/0.1@user/testing -s build_type=Debug
> …
```

Outline

- Introduction
- Consume Conan packages
- Create Conan packages
- Uploading packages to Artifactory
- **Build configuration & cross-build**
- Requirements
- Hooks and Conan configuration
- Versioning
- Jenkins Artifactory Conan CI

Options

Conan allows to build/reuse packages with different configurations:

Settings

- Different build_type
- Different compiler versions
- Different compilers
- Cross building to a different architecture…

Options

- Different options, (shared, static, active FPU, etc)

Exercise 14 – Using options for shared/static

```
class HelloConan(ConanFile):
    name = "hello"
    version = "0.1"
    settings = "os", "compiler", "arch"
    generators = "cmake"
    options = {"shared": [True, False]}
    default_options = "shared=False"
```

```
$ cd training/create_sources
$ conan create . user/testing -o hello:shared=True
$ conan create . user/testing -o hello:shared=True -s build_type=Debug

$ ../catchup.sh # option 14
```

Exercise 15 – Custom option “language”

```
$ cd training/create_options
```

src/hello.cpp

```cpp
void hello(){
    #if GREET_LANGUAGE == 1
        #ifdef NDEBUG
        std::cout << "Hello World Release!" <<std::endl;
        #else
        std::cout << "Hello World Debug!" <<std::endl;
        #endif
    #else
        #ifdef NDEBUG
        std::cout << "HOLA MUNDO Release!" <<std::endl;
        #else
        std::cout << "HOLA MUNDO Debug!" <<std::endl;
        #endif
    #endif
}
```

src/CMakeLists.txt

```cmake
cmake_minimum_required(VERSION 2.8)
project(MyHello CXX)

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

add_library(hello hello.cpp)
target_compile_definitions(hello PRIVATE
      GREET_LANGUAGE=${GREET_LANGUAGE})
```

Exercise 15 – Custom option “language”

```
$ cd training/create_options
```

conanfile.py

```python
class GreetConan(ConanFile):
    name = "greet"
    version = "0.1"
    settings = "os", "compiler", "build_type", "arch"
    options = {"language": ["English", "Spanish"]}
    default_options = "language=English"

    def build(self):
        cmake = CMake(self)
        if self.options.language == "English":
            cmake.definitions["GREET_LANGUAGE"] = 1
        else:
            cmake.definitions["GREET_LANGUAGE"] = 0
        cmake.configure(source_folder="src")
        cmake.build()
```

Exercise 15 – Custom option “language”

```
$ conan create . user/testing -o greet:language=English
$ conan create . user/testing -o greet:language=Spanish


                                        $ ../catchup.sh # option 15
```

Exercise 15 – Errors in configuration

```
$ conan create . user/testing -o greet:language=Italian  # Error
# and for settings?
$ conan create . user/testing -s compiler=unknown # Error
$ conan create . user/testing -s compiler.version=200 # Error
```

Conan settings

```
$ vim ~/.conan/settings.yml
```

```
os:
    Windows:
    Linux:
    Macos:
arch: [x86, x86_64, ppc32be, armv4, ..., asm.js, wasm, sh4le]
compiler:
    gcc:
        version: ["4.1", "4.4", "4.5", ...,"9", "9.1"]
        libcxx: [libstdc++, libstdc++11]
    Visual Studio:
        runtime: [MD, MT, MTd, MDd]
        version: ["8", "9", "10", "11", "12", "14", "15", "16"]
        toolset: [None, v90, v100, v110, ... v142]
    clang:
        version: ["3.3", "3.4", "3.5", ..., "7.0","8"]
```

Custom Conan settings

```
os:
    Windows:
    Linux:
      distro: [None, RHEL6, RHEL7, Centos]
    Macos:
arch: [x86, x86_64, ppc32be, armv4, ..., asm.js, wasm, sh4le]
compiler:
    gcc:
        version: ["4.1", "4.4", "4.5", ...,"9", "9.1"]
        libcxx: [libstdc++, libstdc++11]
    Visual Studio:
        runtime: [MD, MT, MTd, MDd]
        version: ["8", "9", "10", "11", "12", "14", "15", "16"]
        toolset: [None, v90, v100, v110, ... v142]
    clang:
        version: ["3.3", "3.4", "3.5", ..., "7.0","8"]
```

Conan profiles

Conan allows to build/reuse packages with different configurations:

- Different build_type
- Different compiler versions
- Different compilers
- Cross building to a different architecture…
- Different options, (shared, static, active FPU, etc)

```
conan install . -s compiler=gcc -s compiler=4.8 -s
arch=armv7 -s build_type=Release -o
zlib:shared=True
```

?

Conan profiles

- Plain text files with settings + options + environment variables
  - ~/.conan/profiles
- Can be applied to both conan install and conan create
- Can be shared between the team (standard confs for a company)
  - $ conan config install
- Env vars are very useful to enable cross building toolchains (CC, CXX)

```
[settings]
os=Linux
compiler=gcc
compiler.version=4.9
compiler.libcxx=libstdc++
build_type=Debug
arch=armv7

[env]
CC=arm-linux-gnueabihf-gcc
CXX=arm-linux-gnueabihf-g++
```

Conan Profiles

```
$ conan profile list
$ conan profile show default

$ conan create . user/testing
# equal to
$ conan create . user/testing -pr=default
```

Exercise 16 – Cross Build Hello Package to R-PI

```
$ cd cross_build
$ less rpi_armv7
# press "q" to exit less
```

```
[settings]
os=Linux
compiler=gcc
compiler.version=6
compiler.libcxx=libstdc++11
build_type=Release
arch=armv7
os_build=Linux
arch_build=x86_64

[env]
CC=arm-linux-gnueabihf-gcc
CXX=arm-linux-gnueabihf-g++
```

Exercise 16 – Cross Build Hello Package to R-PI

```
$ conan create . user/testing -pr=rpi_armv7
> …
$ conan search
$ conan search hello/0.1@user/testing

                                        $ ../catchup.sh # option 16
```

Profiles: including and variables

```
CROSS_GCC=arm-linux-gnueabihf

include(default)

[settings]
arch=armv7

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

Profiles: per-package settings and env-vars

```
[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
```

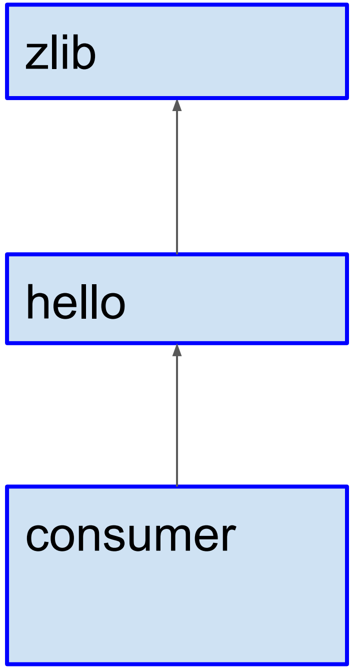
Profiles: composition

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

Outline

- Introduction
- Consume Conan packages
- Create Conan packages
- Uploading packages to Artifactory
- Build configuration & cross-build
- **Requirements**
- Hooks and Conan configuration
- Versioning
- Jenkins Artifactory Conan CI

Exercise 17 - Transitive requirements



Exercise 17 – Transitive requiring zlib

```
$ cd training/requires
```

src/hello.cpp

```cpp
#include <iostream>
#include "hello.h"
#include <zlib.h>

void hello(){
    std::cout << "Hello world!\n";

    char buffer_in [100] = {"some string"};
    char buffer_out [100] = {0};

    z_stream defstream;

    ...

    printf("size: %lu\n", strlen(buffer_out));
```

conanfile.py

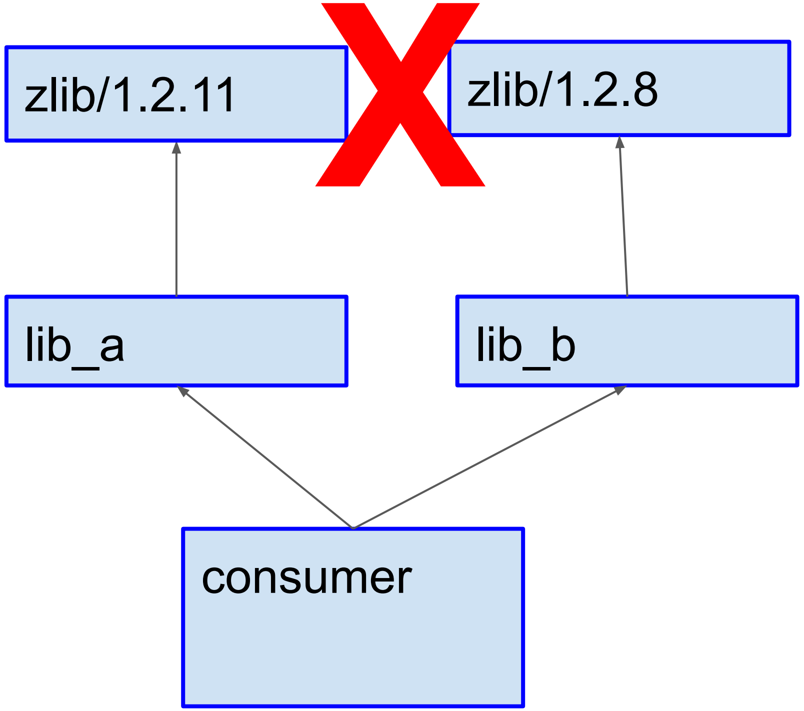
```python
class HelloConan(ConanFile):
    name = "hello"
    version = "0.1"
    settings = "os", "compiler", "arch"
    generators = "cmake"
    exports_sources = "src/*"
    requires = "zlib/1.2.11@conan/stable"
```

Exercise 17 – Transitive requiring zlib

```
$ conan create . user/testing
# What if we try to create the package for RPI?
$ conan create . user/testing -pr=../cross_build/rpi_armv7 # Error
$ conan create . user/testing -pr=../cross_build/rpi_armv7 --build=missing
```

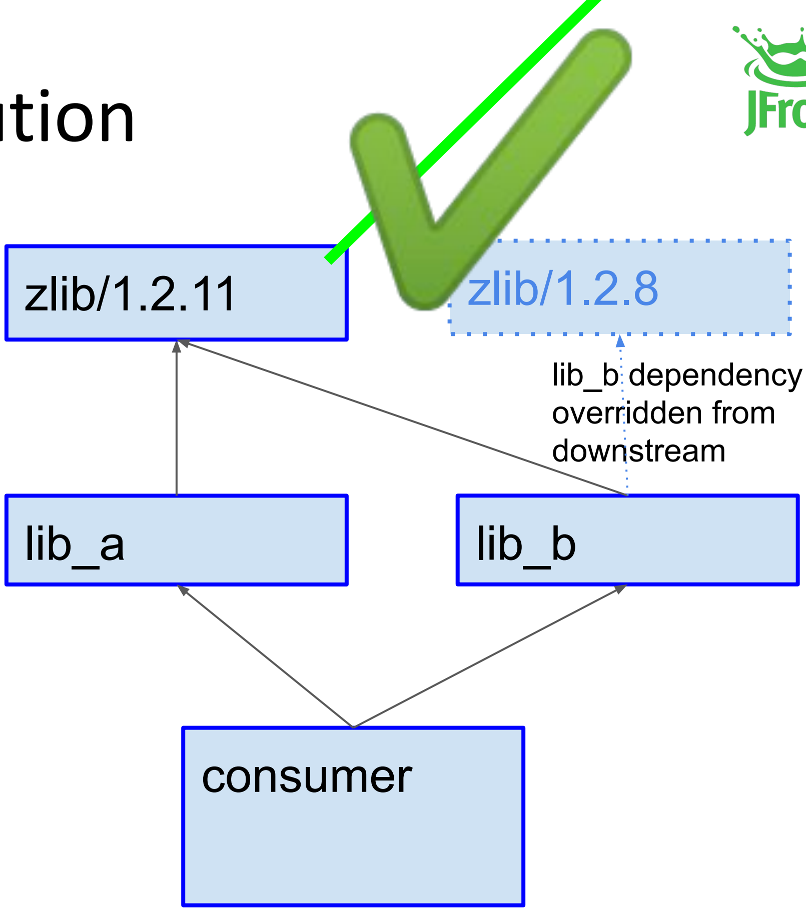
Exercise 18 - Conflicts

```
$ cd requires_conflict
$ conan create lib_a user/testing
$ conan create lib_b user/testing
$ conan install . # Error
```



Exercise 18 - Conflict resolution

```
# Edit consumer conanfile.txt
# add zlib/1.2.11 as [requires]
$ conan install .
```

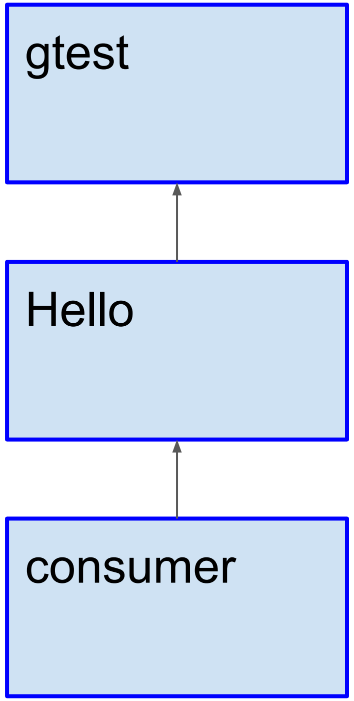


Conditional requirements

conanfile.py

```
class GreetConan(ConanFile):
    name = "greet"
    version = "0.1"
    settings = "os", "compiler", "build_type", "arch"
    options = {"use_ssl": [True, False]}
    default_options = "use_ssl=False"

    def requirements(self):
        if self.options.use_ssl:
            self.requires("openssl/1.0.2a@conan/stable")
```

Exercise 19 – Unit Tests with gtest



Exercise 19 – Unit Tests with gtest

```
$ cd training/gtest/package
```

test.cpp

```cpp
#include <gtest/gtest.h>
#include "hello.h"

TEST(SalutationTest, Static) {
  EXPECT_EQ(string("Hello World!"), message());
}
```

Exercise 19 – Unit Tests with gtest

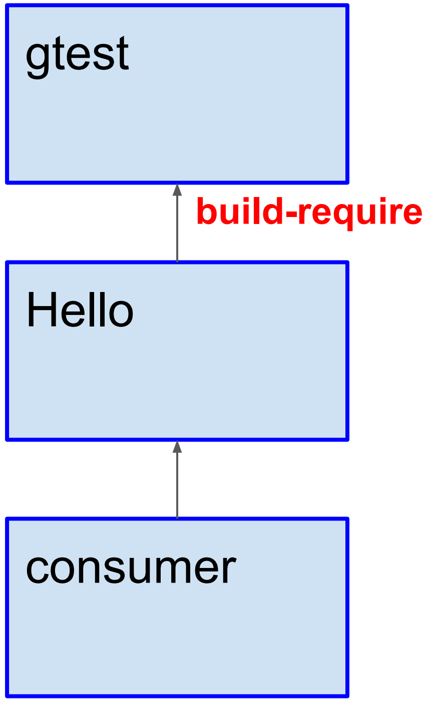
conanfile.py

```
class HelloConan(ConanFile):
    name = "hello"
    version = "0.1"
    settings = "os", "compiler", "build_type", "arch"

    requires = "gtest/1.8.0@bincrafters/stable"
    default_options = "gtest:shared=False"

    def build(self):
        cmake = CMake(self)
        cmake.configure()
        cmake.build()
        self.run("bin/runUnitTests")
```

Exercise 19 – Unit Tests with gtest

```
# search in conan-center for gtest package
$ conan create . user/testing
# Check dependencies
$ cd ../consumer
$ conan install .
# check dependencies (gtest installed!)
```

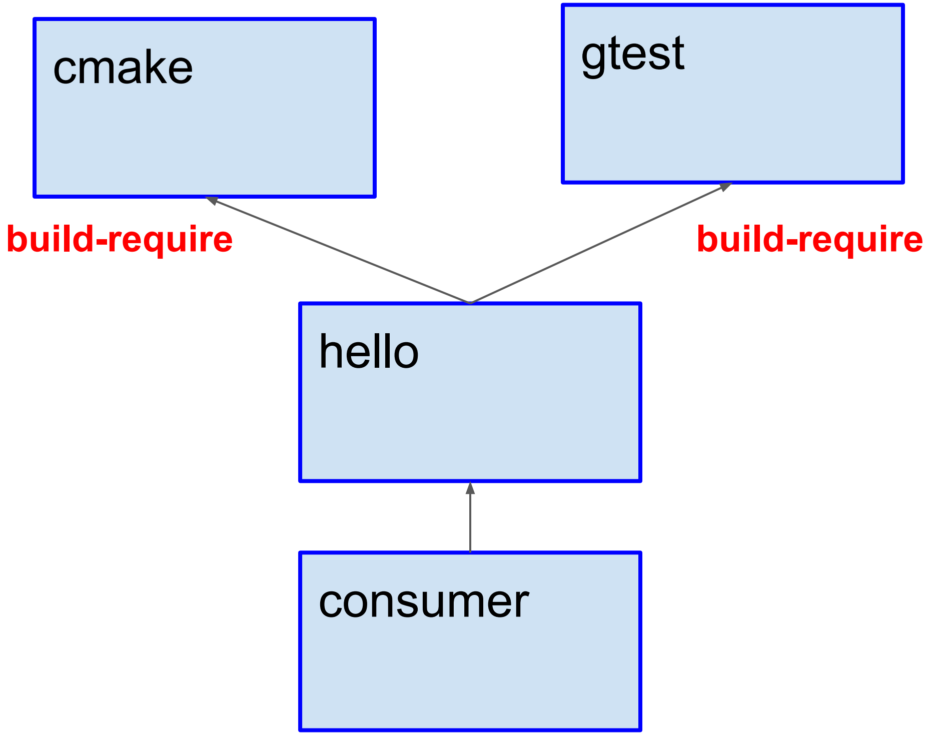
Exercise 20 – Unit Tests with gtest (build-require)



Exercise 20 – Unit Tests w. gtest (build-require)

```
$ cd ../package
# change “requires” □ “build_requires”
$ conan create . user/testing
$ cd ../consumer
$ conan install .
# check dependencies
```

Exercise 21 – CMake as build-require



Exercise 21 – CMake as build-require

conanfile.py

```
class HelloConan(ConanFile):
    name = "hello"
    version = "0.1"
    settings = "os", "compiler", "build_type", "arch"
    generators = "cmake"
    exports_sources = "*"
    build_requires = "gtest/1.8.0@bincrafters/stable",
                     "cmake/3.8@..."
```

Exercise 21 – CMake from build_require

```
$ cmake --version
# check line in CMakeLists:
  message(STATUS "CMAKE VERSION ${CMAKE_VERSION}")
$ conan create . user/testing
# search for a “cmake” package in conan-center
$ vim myprofile
```

Exercise 21 – CMake from build_require

```
myprofile

include(default)

[build_requires]
cmake_installer/3.3.2@conan/stable
```

```
$ conan create . user/testing –pr=myprofile
# Check cmake version!
$ cmake --version
```

A few notes about build_requires

- They shouldn’t change the binary
  - They are not taken into account in the package ID
- Use them for tools:
  - Build tools, like cmake.
    - E.g. OpenSSL in Windows build-requires Nasm and Strawberry Perl
  - Testing frameworks
- Use them in profiles for common things (cmake)
- Use them in recipes for specific, and package specific things (testing framework)

Directly installing packages & virtualenvs

```
$ conan install cmake_installer/3.3.2@conan/stable -g virtualrunenv
$ cmake --version
$ source activate_run.sh
$ cmake --version
```

Exercise 22 - Python requires (mytools)

```
$ cd training/python_requires/mytools
$ conan export . user/testing
```

conanfile.py

```
from conans import ConanFile

def mymsg(conanfile):
    print("MyTool working cool message!!! %s" % conanfile.name)

class ToolConan(ConanFile):
    name = "mytools"
    version = "0.1"
```

Exercise 22 - Python requires (reuse)

```
$ cd training/python_requires/consumer
```

conanfile.py

```python
from conans import ConanFile, python_requires

mytools = python_requires("mytools/0.1@user/testing")

class ConsumerConan(ConanFile):
    settings = "os", "compiler", "build_type", "arch"

    def build(self):
        mytools.mymsg(self)
```

Exercise 22 - Python requires (reuse)

```
$  conan create . consumer/0.1@user/testing
> … MyTool working cool message!!!
```

NOTES

- python-requires DO NOT have binary packages, only python code
- They do not affect the package-ID
- python-requires can have dependencies to other python-requires (keep minimum)
- A recipe can have multiple python requires
- They might contain other files (source file, build scripts)

Python requires (inheritance)

```
from conans import ConanFile


class BaseConanFile(ConanFile):
     ...
    def build(self):
        ...
   def package(self):
        ...
   def package_info(self):
```

Python requires (inheritance II)

```
from conans import ConanFile, python_requires
mytools = python_requires("mytools/0.1@user/testing")


class Pkg(mytools.BaseConanFile):
    # inherits the source(), build()...
```

Outline

- Introduction
- Consume Conan packages
- Create Conan packages
- Uploading packages to Artifactory
- Build configuration & cross-build
- Requirements
- **Hooks and Conan configuration**
- Versioning
- Jenkins Artifactory Conan CI

Hooks

- Hooks are users extensions, written in python,at some points:
  - pre_build(), post_build(), pre_package(), post_package()...
- Should be orthogonal to recipes: custom checks, auxiliary logic.
- Stored in cache: <userhome>/.conan/hooks
- Activated in: <userhome>/.conan/conan.conf

Hooks

```
$  vim myconfig/hooks/check_name.py
```

```python
def pre_export(output, conanfile, conanfile_path,
               reference, **kwargs):
   ref = str(reference)
   if ref.lower() != ref:
       raise Exception("%s should be lowercase" % ref)
```

Hooks: how to activate them

```
# Copy hook in <username>/.conan/conan.conf
$ cp myconfig/hooks/check_name ~/.conan/hooks
# Activate in conan.conf
$ vim ~/.conan/conan.conf
[hooks]
check_name
```

```
$ conan new Hello/0.1
$ conan create . user/testing # Error
```

conan config install

- Command that can install/update in cache:
  - Add/update: hooks, profiles
  - Update: settings.yml, remotes.txt
  - Add any other file (pylintrc)
- From:
  - A git repo (master branch)
  - A remote http zip file
  - A local zip file
  - A local folder

Exercise 23 - conan config install & hooks

```
$ conan config install myconfig # can be URL, git

$ cd hooks
$ conan new Hello-Pkg/0.1 -s
$ conan export . user/testing # Error
$ conan new hello-pkg/0.1 -s
$ conan export . user/testing # OK
```

Exercise 23 - conan config install & hooks

- Modify hook to forbid “-” (recommend “_”), in “myconfig” configuration
- Do “conan config install” (try without arguments)
- Try to create a package with “-”

```
# goal
$ conan new hello-pkg/0.1 -s
$ conan export . user/testing # Error
$ conan new hello_pkg/0.1 -s
$ conan export . user/testing # OK
```

Outline

- Introduction
- Consume Conan packages
- Create Conan packages
- Uploading packages to Artifactory
- Build configuration & cross-build
- Requirements
- Hooks and Conan configuration
- **Versioning**
- Jenkins Artifactory Conan CI

Approaches to versioning

- Bump version (semver):
  - 1.2.3->1.2.4
  - 2.8.12->3.0.0
  - What if you are packaging Boost 1.64, and need to do a change to the recipe?
    - 1.64.1? Mismatch to the original Boost version
  - Versions might use version ranges requirements
- Revisions:
  - pkg/version@user/channel#revision
  - revision is internal, automatic (hash)

Exercise 24 - Version ranges

```
$  cd training/version_ranges
```

conanfile.py

```python
class ChatConan(ConanFile):
    name = "chat"
    version = "0.1"
    ..
    requires = "hello/[>0.0 <1.0]@user/testing"
```

chat.cpp

```cpp
void chat(){
    hello();
    hello();
    hello();
}
```

Exercise 24 - Version ranges

```
$ conan create hello1 user/testing
$ conan create chat user/testing
```

Exercise 24 - Version ranges

```
# generate a new hello/0.2 version (check hello.cpp)
$  conan create hello2 user/testing

# the chat package will use it because it is inside its
valid range
$  conan create chat user/testing

                                              # catchup.sh
```

Version ranges

```
$ conan install "hello/[>0.0 <1.0]@user/testing"
$ conan install "hello/[*]@user/testing"
$ conan install "hello/[~1.1]@user/testing"
```

Lockfiles

- A snapshot of a dependency graph at a given time.
- Can be use to reconstruct the exact same graph of dependencies

Exercise 25 - Lockfiles

```
$  cd training/version_ranges
# make sure we remove hello/0.2 by now
$  conan remove hello/0.2* -f

# will generate a conan.lock file
$  conan graph lock chat
# inspect conan.lock, what is in it?
# Create a new hello/0.2 version
$  conan create hello2 user/testing
```

Exercise 25 - Lockfiles

```
# this will use the new hello/0.2, it is in the range
$ conan create chat user/testing

# Using the lockfile the chat package will NOT use 0.2
it is locked to 0.1
# Reproducible dependency graph!
$ conan create chat user/testing --lockfile
```

Exercise 26 - Package revisions (creating)

```
$ conan config set general.revisions_enabled=True
# check the conan.conf

$ mkdir revisions && cd revisions
$ conan remove hello* -f # remove previous
$ conan new hello/0.1 -s
$ conan create . user/testing
$ conan create . user/testing -s build_type=Debug
$ conan upload hello* --all -r=artifactory --confirm
# check in Artifactory
```

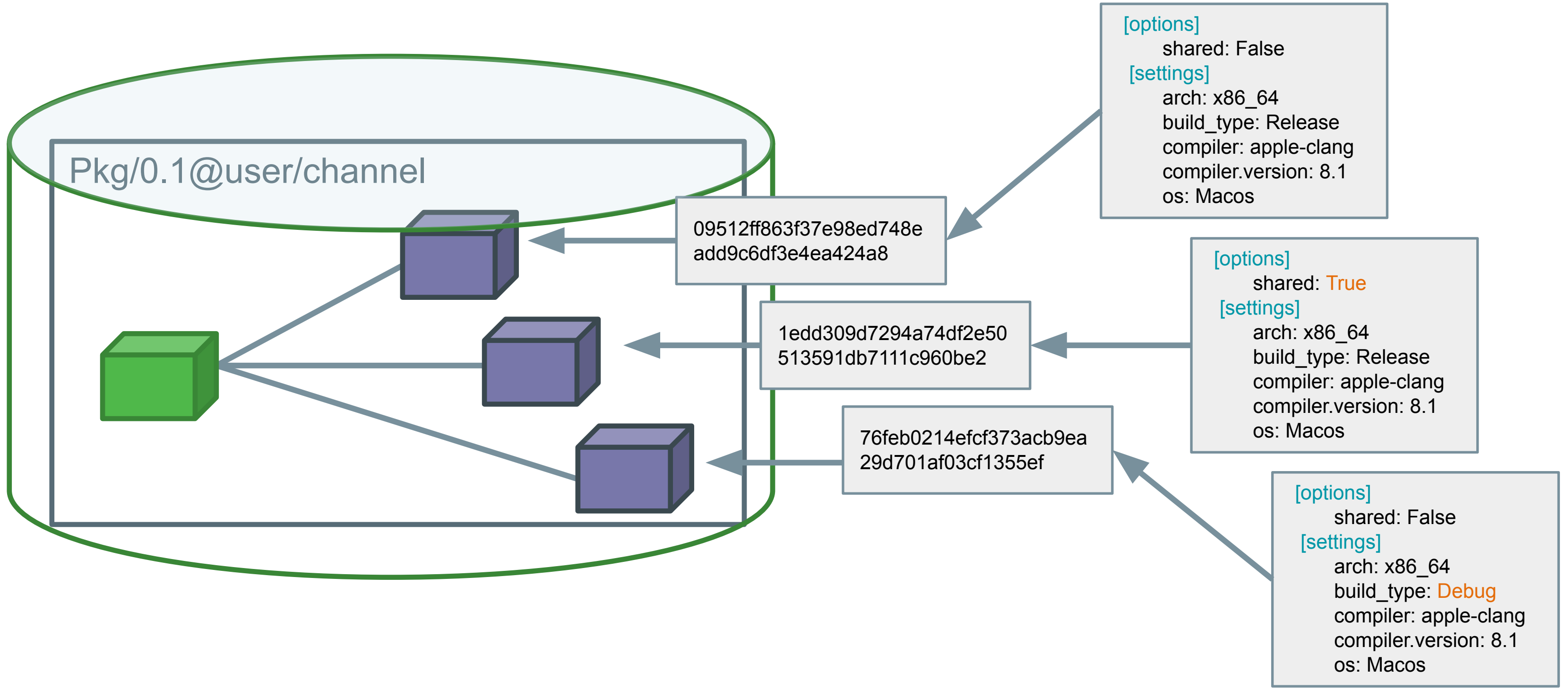
Exercise 26 - Package revisions (creating)

```
$ echo "#comment" >> conanfile.py
$ conan create . user/testing
$ conan create . user/testing -s build_type=Debug
$ conan upload hello* --all -r=artifactory --confirm
# check in Artifactory
```

Exercise 26 - Package revisions (consuming)

```
$ conan remove hello* -f
$ conan install hello/0.1@user/testing
# By default latest revision
$ conan remove hello* -f
$ conan install hello/0.1@user/testing#<revision>
```

Binary Management



```
09512ff863f37e98ed748e
add9c6df3e4ea424a8
```

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

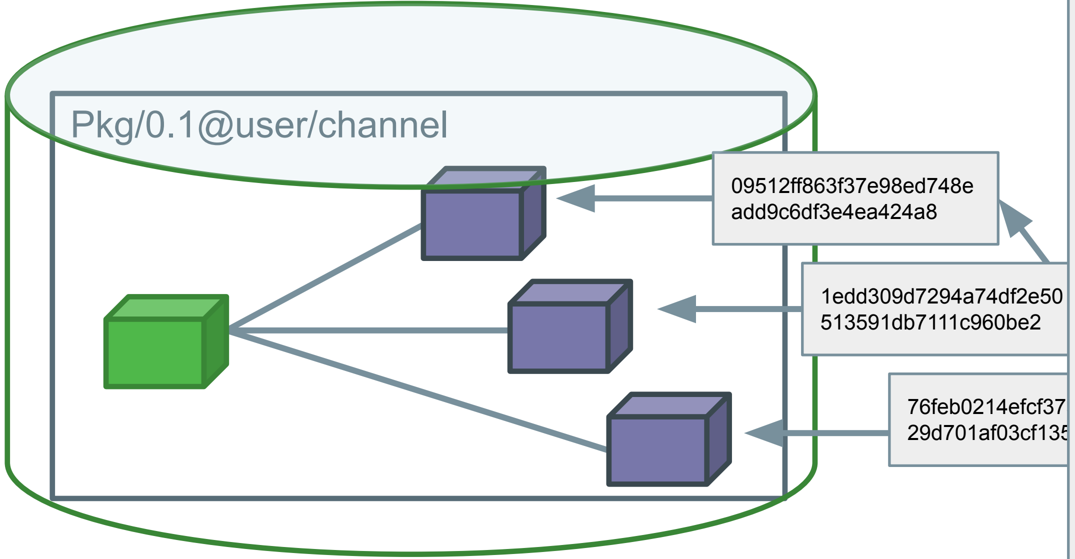
```
1edd309d7294a74df2e50
513591db7111c960be2
```

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

```
76feb0214efcf373acb9ea
29d701af03cf1355ef
```

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

Binary Management



```
[options]
    shared: False
[settings]
    arch: x86_64
    build_type: Release
    compiler: apple-clang
    compiler.version: 8.1
    os: Macos
[requires]
    zlib/1.Y.Z
    poco/2.Y.Z
```

os: Macos

package_id()

conanfile.py

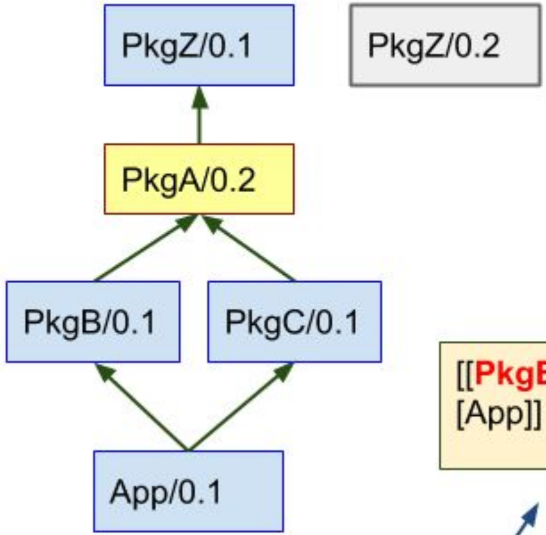
```
class Pkg(ConanFile):

    ..
    def package_id(self):
        # apply full_package_mode for all the dependencies
        self.info.requires.full_package_mode()
        # use full_package_mode just for MyOtherLib
        self.info.requires["MyOtherLib"].full_package_mode()
```

package_id()

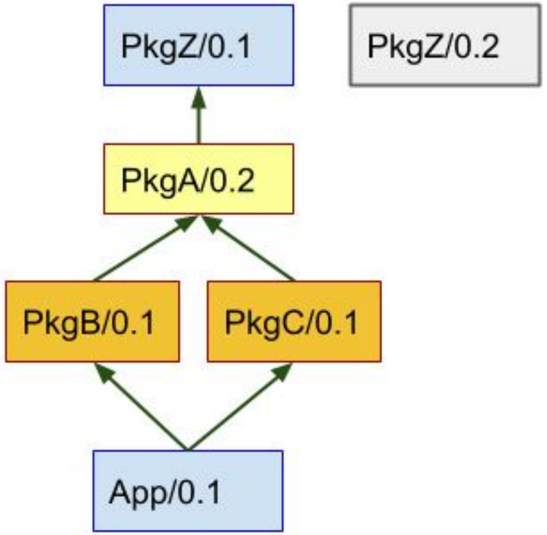
conan.conf

```
[general]
default_package_id_mode=full_package_mode
```

Package-IDs + Revisions + Lockfiles: CI for C++ at scale



[[PkgB/0.1, PkgC/0.1],
[App]]



```
$ conan graph build-order ./release
--json=bo.json --build=missing
```

In parallel:

```
conan install PkgB/0.1  --build=PkgB/0.1 --lockfile
conan install PkgC/0.1  --build=PkgC/0.1 --lockfile
```

Outline

- Introduction
- Consume Conan packages
- Create Conan packages
- Uploading packages to Artifactory
- Build configuration & cross-build
- Requirements
- Hooks and Conan configuration
- Versioning
- **Jenkins Artifactory Conan CI**

Exercise 27 – Hello JSON (Part I PicoJson)

- Try to package the open source library Pico JSON: https://github.com/kazuho/picojson.git
- Go to pico_json folder,
  use the **example.cpp** for your test_package
- Hint: Use “conan new --help”

Exercise 27 – Hello JSON (Part II Hello)

- Use an option in the “hello” package, by default is False
- If the option is False, “hello” package will say Hello World as always
- If the option is True “hello” will also use the PicoJson library to do something.
- PicoJson has to be required only if the option is True

Exercise – Jenkins CI

```
$ docker exec -it jenkins /bin/bash
$ cd /var/lib/jenkins # We are going to create a new repo
$ mkdir hello && cd hello
$ conan new hello/0.1 –s –t   # lowercase!
$ git init .
$ git checkout -b release/0.1
$ git add .
$ git commit -m “initial release”
```

Exercise – Jenkins CI

```
# Go to Jenkins (IP:8083)
```

Exercise – Jenkins CI

Configure Jenkins Job:

- New Item -> Multibranch Pipeline -> Give Name (conan-hello) -> OK
- Branch sources -> Add source -> Enter path to repo "/var/lib/jenkins/hello"
- Scan Multibranch Pipeline Triggers => Check "periodically" => 1 min
- Save button

Then:

- Check build, check logs

Jenkinsfile

```
def artifactory_name = "artifactory-ha"
def artifactory_repo = "myconanrepo"

node {
   def server = Artifactory.server artifactory_name
   def client = Artifactory.newConanClient()
   def serverName = client.remote.add server: server, repo: artifactory_repo
   stage("Get recipe"){
       checkout scm
   }

   stage("Build package"){
       client.run(command: "create . team/stable")
   }

   stage("Upload packages"){
       String command = "upload * --all -r ${serverName} --confirm"
       def b = client.run(command: command)
       server.publishBuildInfo b
   }
}
```

Exercise – Jenkins CI

```
$ wget
https://raw.githubusercontent.com/conan-io/training/master/jenkins/Jenkinsfile
$ git add .
$ git commit -m "Jenkinsfile"
```

Exercise – Jenkins CI

Generate a new package version

- Create new branch “release/0.2”
- Bump the version number in “conanfile.py” (and the .cpp code if you want)
- Commit the changes
- Check CI logs and Artifactory

Exercise – Jenkins CI

Generate revisions of every release version

- Enable revisions in the Jenkinsfile
- Do changes to the source code
- Commit
- Wait for Jenkins to create the revisions
- Check in Artifactory

THANK YOU!

JFrog

CONAN
C/C++ Package manager

Exercise - SCM

```
$ cd
$ cd training/scm
$ vim conanfile.py
```

- NO source() method necessary
- NO exports_sources necessary
- It captures the url & revision
- It does NOT capture the sources
- It can reproduce the build

```
scm = {"type": "git",
        "url": "auto",
        "revision": "auto"}
```

Exercise - SCM

```
$ conan create . user/testing
$ conan get hello/0.1@user/testing
```