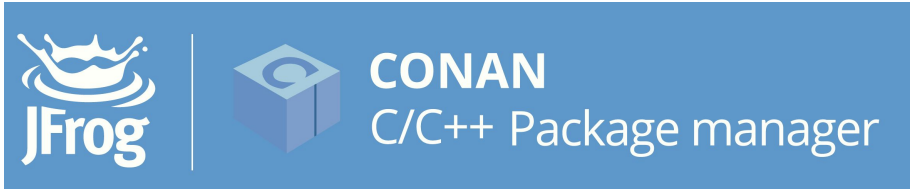


Introduction to Conan C++ Package Manager: Basics

Jacob Lärfors, Consultant and Co-Founder @ Verifa
Diego Rodriguez-Losada, Conan co-founder @ JFrog

Copyright @ 2020 JFrog - All rights reserved

Coaches

Jacob Lärfors, CTO

verifa

Diego Rodriguez-Losada, Conan co-founder

JFrog

Technical Assistants

JFrog

- **Daniel Manzaneque**
- **Javier Garcia**
- **Carlos Zoido**

Verifa

- **James Elderfield**

Copyright @ 2020 JFrog - All rights reserved.

Verifa, *Continuous Everything*

Services company specialising in DevOps with a focus on:

- Continuous Integration & Delivery
- Modern Infrastructure
- Analytics
- OSS Compliance

Strong background in the embedded space (medical, automotive, industrial)

Conan was a natural fit to solve CI-related problems, such as:

- Slow builds, due to lots of inefficient rebuilding
- Configuration Management Traceability
- Build environment setup a pain
- Monolithic architectures
- Homegrown, weird/unusual alternative to package management
  - e.g. using shared network drive, rsync, etc.

verifa.io

Copyright @ 2020 JFrog - All rights reserved.

Exercise 0 - 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
--------------------
>
```

Copyright @ 2020 JFrog - All rights reserved.

Outline

- **Introduction**
- Consume Conan packages
- Create Conan packages
- Uploading packages to Artifactory
- Build configuration & cross-build

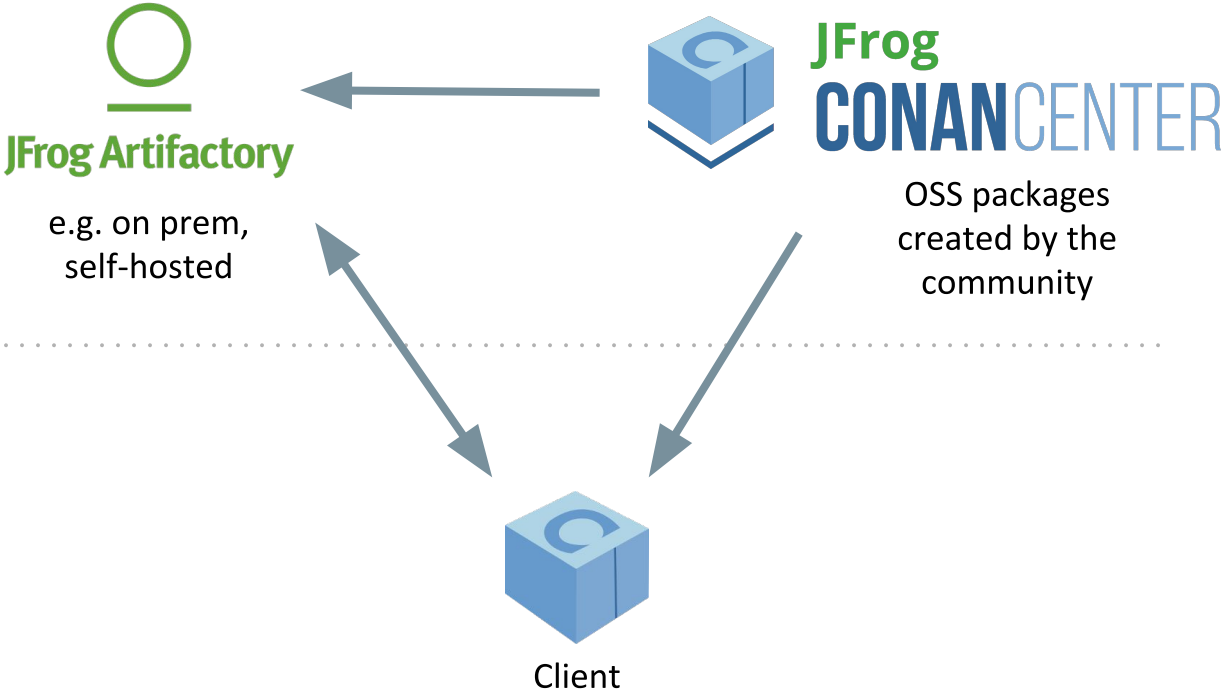
Copyright @ 2020 JFrog - All rights reserved.

Introduction

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

Copyright @ 2020 JFrog - All rights reserved.

Architecture

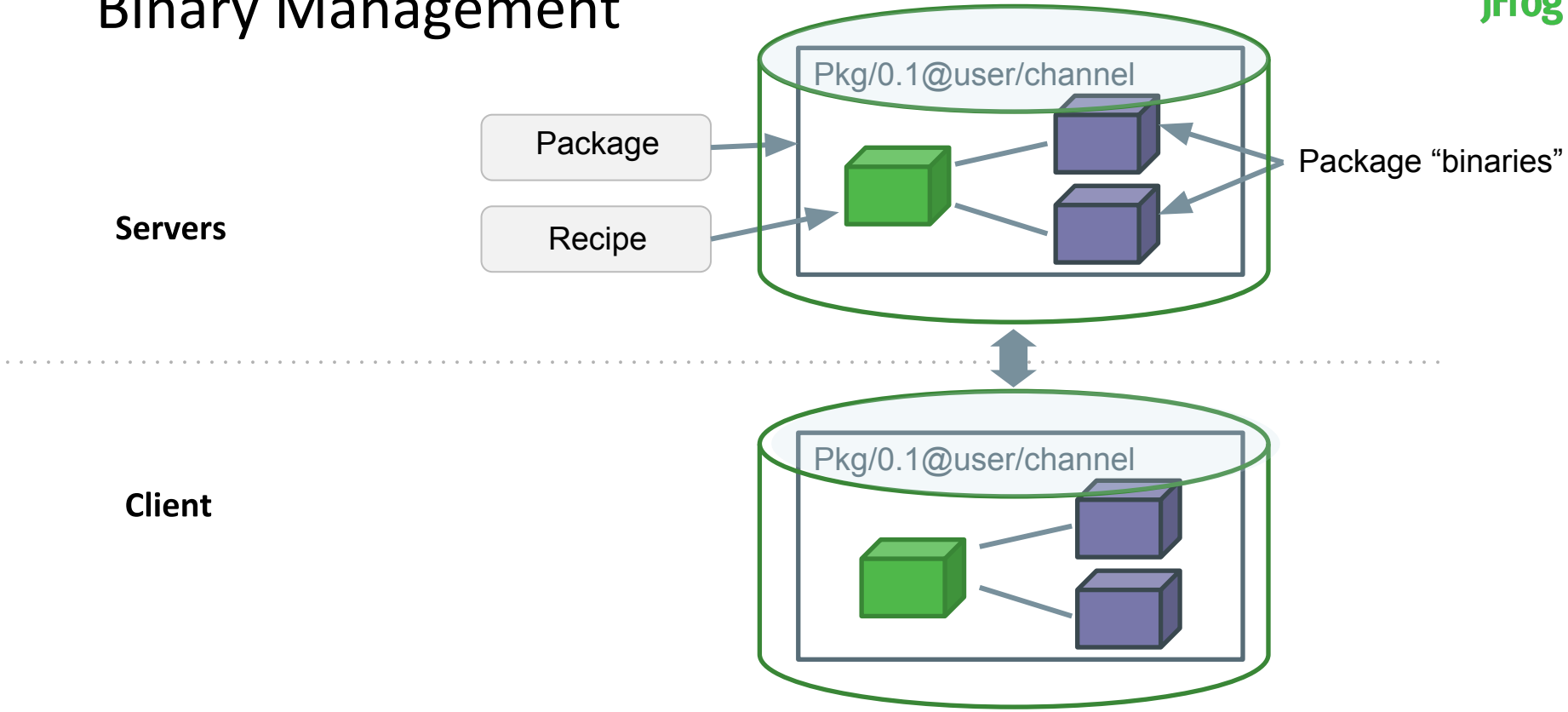


Servers
(artifact storage)

Developer
machine / CI

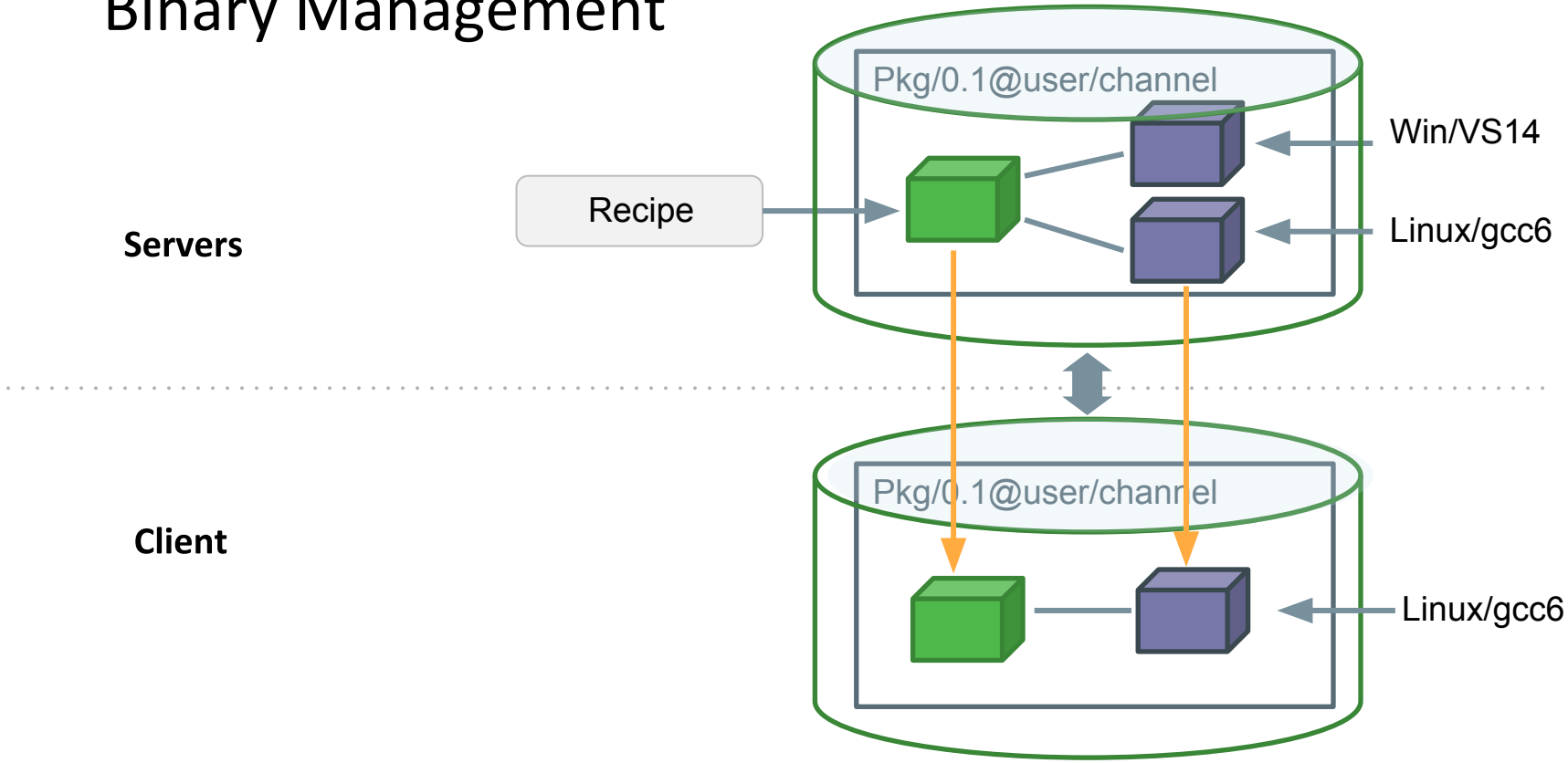
Copyright @ 2020 JFrog - All rights reserved.

Binary Management



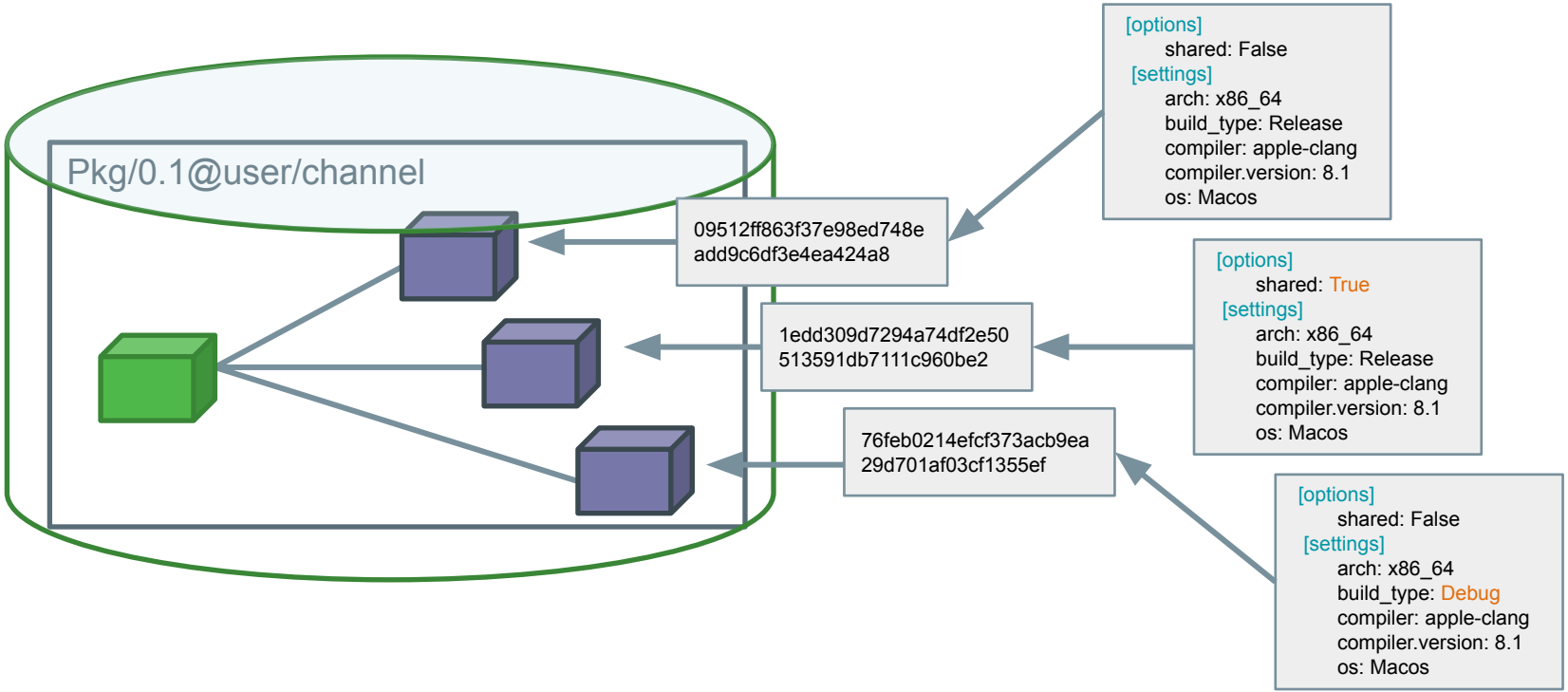
Copyright @ 2020 JFrog - All rights reserved.

Binary Management



Copyright @ 2020 JFrog - All rights reserved.

Binary Management



Copyright @ 2020 JFrog - All rights reserved.

Here be dragons

- The “catchup.sh” script to be used if lost
- There are bugs, on purpose
- They are solved by the “catchup.sh”

Copyright @ 2020 JFrog - All rights reserved.

Outline

- Introduction
- **Consume Conan packages**
- Create Conan packages
- Uploading packages to Artifactory
- Build configuration & cross-build

Copyright @ 2020 JFrog - All rights reserved.

Exercise 1 - Consume with CMake

Goal:

- The goal of this exercise is to use some Conan packages using CMake

Task:

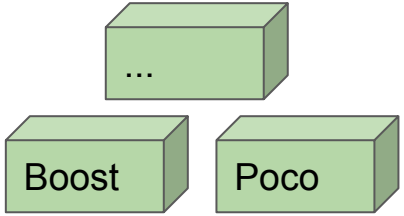
- Source code in the **training/consumer** directory
- Build and Run the program

Success:

- The Timer application runs

Copyright @ 2020 JFrog - All rights reserved.

Exercise 1 - ConanCenter



JFrog CONANCENTER

OSS packages created by the community

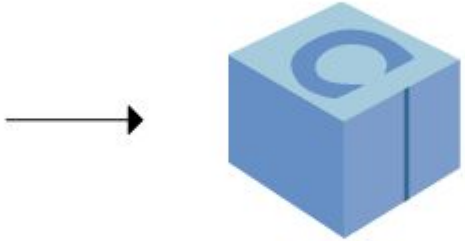
Servers
(artifact storage)

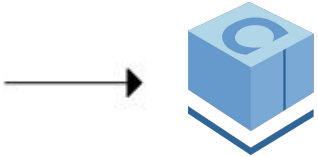
Developer
machine / CI

Client

Copyright @ 2020 JFrog - All rights reserved.

ConanCenter build service for contributors





Contributors do PR to the repository with package recipes

https://github.com/conan-io/conan-center-index

Builds +100 binaries for every package version

ConanCenter serves the packages

https://conan.io/center/

Copyright @ 2020 JFrog - All rights reserved.

Exercise 1 - Consume with CMake

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

timer.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/1.72.0
poco/1.9.4

[generators]
cmake

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

CMakeLists.txt

```
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(TARGETS)

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

Copyright @ 2020 JFrog - All rights reserved.

Exercise 1 - Consume with CMake

3:00

```
$ 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 1

Copyright @ 2020 JFrog - All rights reserved.

Exercise 2 - Consume (debug mode)

Goal:

- Consume debug builds for the Timer application

Task:

- Source code in the **training/consumer** directory
- Conan install with build_type=Debug
- Build with debug and run

Success:

- The Timer application runs

Copyright @ 2020 JFrog - All rights reserved.

Exercise 2 - Installed Packages (search)

```
$ conan search
$ conan search zlib/1.2.11@
...
# check build_type: Release
```

Copyright @ 2020 JFrog - All rights reserved.

Exercise 2 - Consume (debug mode)

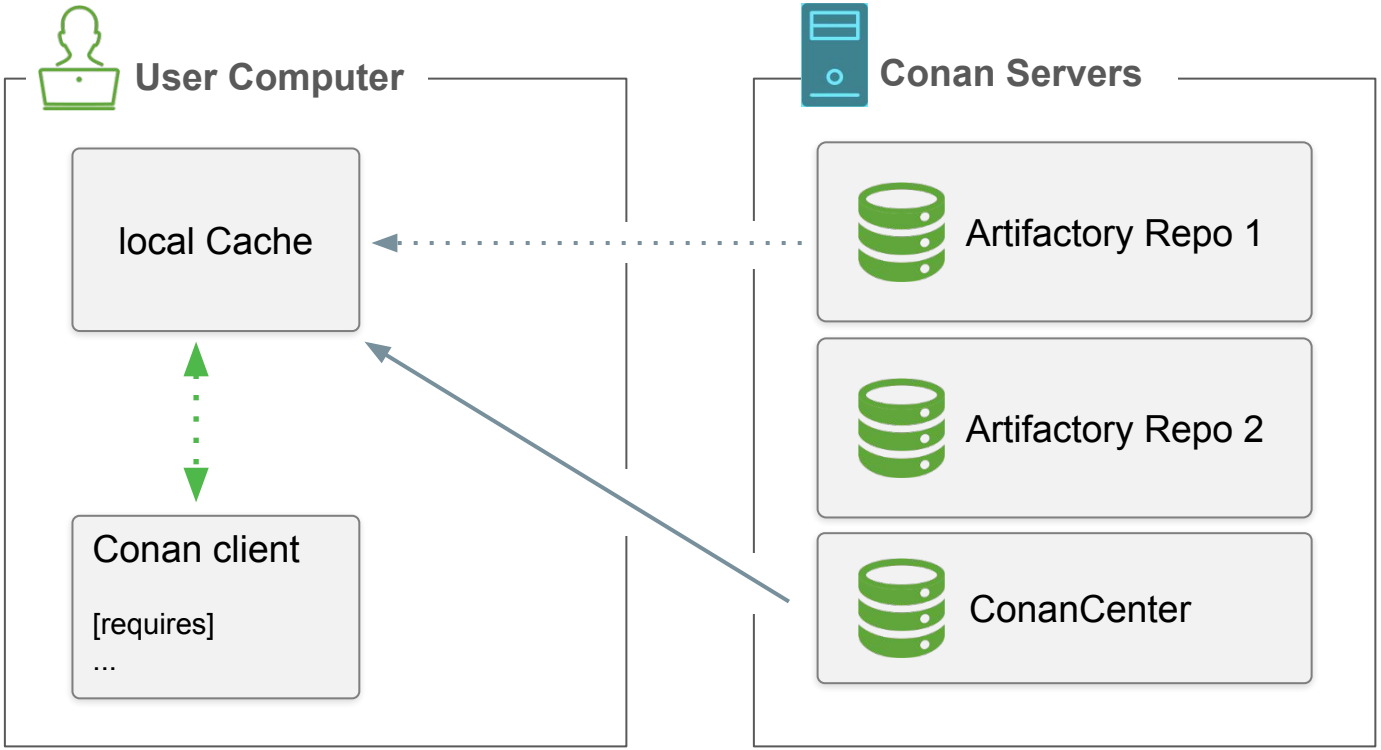
3:00

```
$ 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@
# check the two packages with different build_type
```

$ ../catchup.sh # option 2

Copyright @ 2020 JFrog - All rights reserved.

Exercise 2 - How Conan Installs Packages



Copyright @ 2020 JFrog - All rights reserved.

Conan Info & Search

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

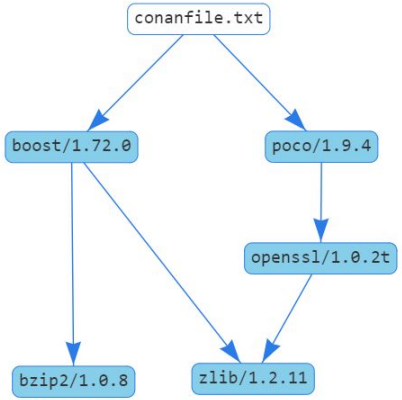
zlib/1.2.11@conan/stable

	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



Copyright @ 2020 JFrog - All rights reserved.

Exercise 3 - Consume (gcc)

Goal:

- Understand generators. There are different generators in Conan
- Let's use plain gcc compiler

Task:

- Source code in the **training/consumer_gcc** directory
- Conan install with ***-g compiler_args*** to specify the generator in cmd-line
- Build and run

Success:

- The Timer application runs

Copyright @ 2020 JFrog - All rights reserved.

Exercise 3 - Consume (gcc generator)

3:00

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

$ ../catchup.sh # option 3

Copyright @ 2020 JFrog - All rights reserved.

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!

Copyright @ 2020 JFrog - All rights reserved.

Exercise 4 - Consume with CMake Find

Goal:

- Let's use CMake again, but the **cmake_find_package** generator
- Useful as nothing specific about Conan in CMakeLists.txt

Task:

- Source code in the **training/consumer_cmake_find** directory
- Same as before… conan install, build and run

Success:

- The Timer application runs

Copyright @ 2020 JFrog - All rights reserved.

Exercise 4 - Consume with CMake find

```
$ cd consumer_cmake_find
```

conanfile.txt

```
[requires]
boost/1.72.0
poco/1.9.4

[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)
```

Copyright @ 2020 JFrog - All rights reserved.

Exercise 4 - 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
>...
```

$ ../catchup.sh # option 4

Copyright @ 2020 JFrog - All rights reserved.

Outline

- Introduction
- Consume Conan packages
- **Create Conan packages**
- Uploading packages to Artifactory
- Build configuration & cross-build

Copyright @ 2020 JFrog - All rights reserved.

Exercise 5 - Create Package (from GitHub source)

Goal:

- Create a Conan package of a “Hello World” C++ library from GitHub
- Use our first **Python conanfile.py recipe**
- Create different binaries (Release/Debug) of the package

Task:

- Empty directory: **training/create**
- Use the “conan new” command which creates the recipe for us
- Use the “conan create” command to create the package for us

Success:

- Conan create succeeds
- Conan **search** to show package binaries

Copyright @ 2020 JFrog - All rights reserved.

Exercise 5 - Create Package (from GitHub source)

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

Copyright @ 2020 JFrog - All rights reserved.

```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"]
```

Copyright @ 2020 JFrog - All rights reserved.

Exercise 5 - user/channel attributes

- Recap on user/channel attributes
  - my_package/0.1@user/channel

- No user/channel for official packages from ConanCenter

- user/channel used to distinguish custom packages such as ones you create, or forked from ConanCenter

- user = company/team/identifier
- channel = convention is testing or stable

Copyright @ 2020 JFrog - All rights reserved.

Exercise 5 - Create Package (from GitHub source)

```
$ 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

Copyright @ 2020 JFrog - All rights reserved.

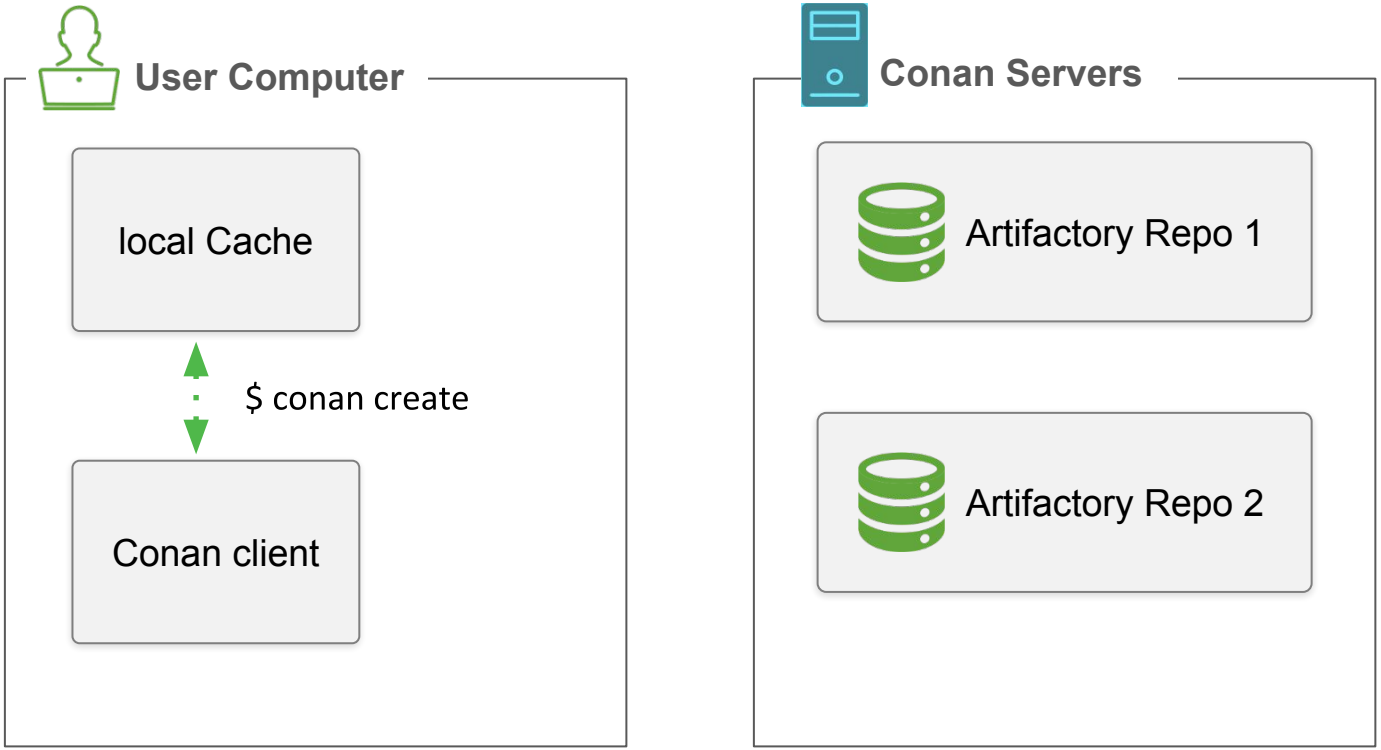
Exercise 5 - Create Package (debug mode)

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

```
$ ../catchup.sh # option 5
```

Copyright @ 2020 JFrog - All rights reserved.

Conan Create is Local



Copyright @ 2020 JFrog - All rights reserved.

Exercise 6 - Consume “hello” package

Goal:

- Consume our “hello” package using our original consumer project

Task:

- Reuse directory: **training/consumer**
- Add a Conan dependency on package hello/0.1@user/testing
- Add the CONAN_PKG to CMakeLists.txt
- Add source code dependency in timer.cpp to call “hello”
- Rebuild and rerun the timer application

Success:

- Timer now says “Hello World Release!”

Copyright @ 2020 JFrog - All rights reserved.

Exercise 6 - Consume “hello” package

```
$ cd consumer
# modify conanfile.txt to account for new dependency
# modify CMakeLists.txt to account for new CONAN_PKG::hello
# modify timer.cpp to include “hello.h” and call the hello() function
# conan install to update dependency graph and conanbuildinfo.cmake
# run cmake, build and run timer
```

Copyright @ 2020 JFrog - All rights reserved.

Exercise 7 - Create & test package

Goal:

- Create a package with a ***test_package***
- ***test_package*** provides a basic test to make sure our package is created and can be consumed

Task:

- Reuse directory: **training/create**
- "***conan new … -t***" command to create the recipe and test_package
- Run conan create

Success:

- Package is created and the test_package is run

Copyright @ 2020 JFrog - All rights reserved.

Exercise 7 - 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();
}
```

Copyright @ 2020 JFrog - All rights reserved.

Exercise 7 - Create & test package

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

Copyright @ 2020 JFrog - All rights reserved.

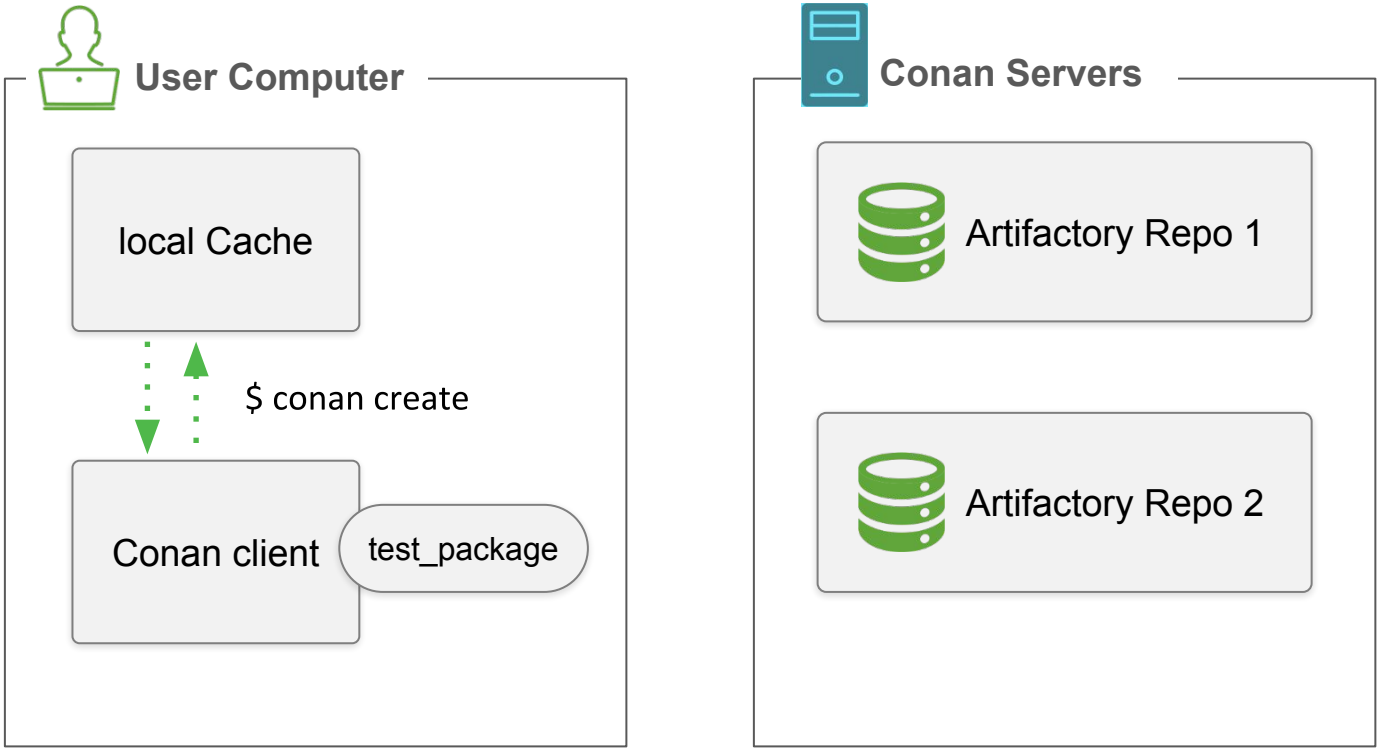
Exercise 7 - Create & test package (debug mode)

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

```
$ ../catchup.sh # option 7
```

Copyright @ 2020 JFrog - All rights reserved.

Conan Create (with test_package) is Local



Copyright @ 2020 JFrog - All rights reserved.

Exercise 8 - Create with source

Goal:

- Create a package where the recipe and source live **together**

Task:

- Empty Directory: **training/create_sources**
- “conan new … -t **-s**”
- Run conan create

Success:

- Package is created using local source

Copyright @ 2020 JFrog - All rights reserved.

Exercise 8 - Create with Source

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

Copyright @ 2020 JFrog - All rights reserved.

```python
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"]
```

Copyright @ 2020 JFrog - All rights reserved.

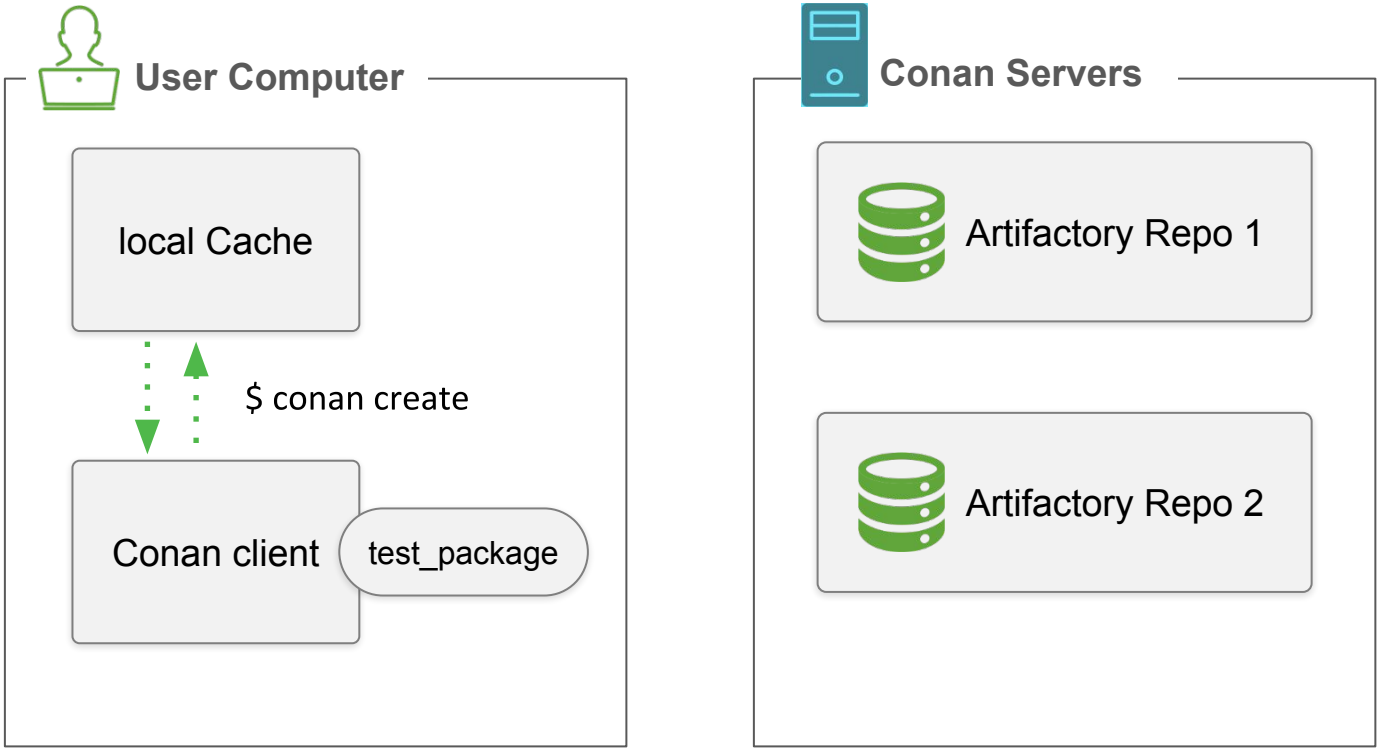
Exercise 8 - Create with Source

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

```
$ ../catchup.sh # option 8
```

Copyright @ 2020 JFrog - All rights reserved.

Conan Create (with test_package) is Local



Copyright @ 2020 JFrog - All rights reserved.

SCM

```
class HelloConan(ConanFile):
    name = "hello"
    version = "0.1"
    settings = "os", "compiler", "build_type", "arch"
    generators = "cmake"
    # NO exports_sources
    scm =  {
        "type": "git",
        "url": "auto",
        "revision": "auto",
    }

    # NO SOURCE METHOD

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

Copyright @ 2020 JFrog - All rights reserved.

Outline

- Introduction
- Consume Conan packages
- Create Conan packages
- **Uploading packages to Artifactory**
- Build configuration & cross-build

Copyright @ 2020 JFrog - All rights reserved.

Exercise 9 - Upload to Artifactory

Goal:

- Upload a package to a *remote*, i.e. a Conan repository in Artifactory

Task:

- Create a Conan repository in Artifactory
- Upload a package to that repository

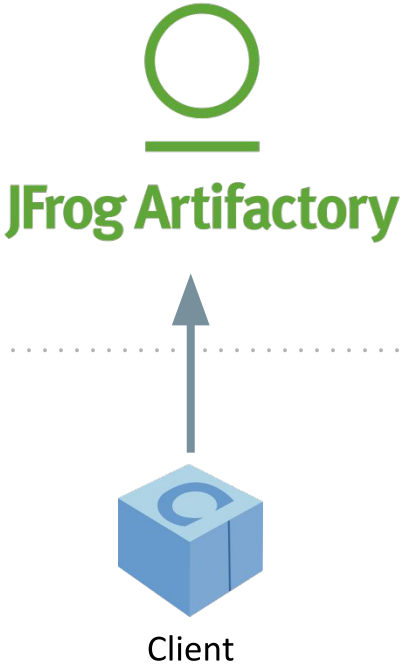
Success:

- Package exists in Artifactory

Copyright @ 2020 JFrog - All rights reserved.

Exercise 9 - Upload to Artifactory

Servers
(artifact storage)



Developer
machine / CI

Copyright @ 2020 JFrog - All rights reserved.

Conan Remotes

```
$ conan remote list
```

Copyright @ 2020 JFrog - All rights reserved.

Artifactory (create new Conan repo)

1 Administration
2 Repositories

Not secure | 34.67.115.149:8082/ui/admin/repositories/local

JFrog Platform

Search admin resources

Welcome, admin

Administration

Repositories

Repositories

Layouts

Identity and Access

Security

Licenses

General

Enterprise Plus trial license
7.2.1 rev 70201900 Licensed
to JFrog TEST
© Copyright 2020 JFrog Ltd

Repositories

Local | Remote | Virtual | Distribution

3. New Local Repository

9 Repositories

Filter

Repository Key	Type	Recalculate Index	Replications ...
automation-mvn-sol-snapshot-local	Maven		
automation-mvn-solution-local	Maven		
conan-local	Conan		
docker-local	Docker		

Copyright @ 2020 JFrog - All rights reserved.

Artifactory (create new Conan repo)

Not secure | 34.67.115.149:8082/ui/admin/repositories/local/new

JFrog Platform

Search admin resources

Welcome, admin

Administration

Repositories

Repositories

Layouts

Identity and Access

Security

Licenses

General

Enterprise Plus trial license 7.2.1 rev 70201900 Licensed to JFrog TEST
© Copyright 2020 JFrog Ltd

New Local Repository

Basic | Advanced | Replications

Package Type *

Conan

4. Select Conan type

5. Name it "myconanrepo"

Repository Key *

myconanrepo

General

Repository Layout

Include / Exclude Patterns

Include Patterns

6. Save & Finish

Cancel

Save & Finish

Copyright @ 2020 JFrog - All rights reserved.

Artifactory (get repo URL)

7. Application
8. Artifactory
9. Artifacts

JFrog Platform | Artifacts | Search Artifacts | Welcome, admin

Application
Dashboard
Artifactory
Packages
Builds
Artifacts
Distribution
Pipelines

Enterprise Plus trial license 7.2.1 rev 70201900 Licensed to JFrog TEST
© Copyright 2020 JFrog Ltd

Artifact Repository Browser | Set Me Up | Deploy

Tree | Simple | ıanrepo

- conan-local
- docker-local
- generic-local
- libs-release-local
- libs-snapshot-local
- libs-staging-local
- **myconanrepo**
- pypi-local
- bintray-docker-remote
- conan-remote
- docker-remote
- jcenter
- pypi-remote
- Trash Can

10. click “myconanrepo”

myconanrepo | Actions

11. Click Set Me Up

General | Effective Permissions | Properties

Info

Name:	myconanrepo
Repository Path:	myconanrepo/
URL to file:	
Package Type:	Conan
Repository Layout:	conan-default
Artifact Count / Size:	Show
Created:	26-03-20 22:50:1 +00:00

Copyright @ 2020 JFrog - All rights reserved.

Artifactory (get repo URL)

JFrog Platform | Artifacts | Search Artifacts | Welcome, admin

Application
- Dashboard
- Artifactory
  - Packages
  - Builds
  - Artifacts
- Distribution
- Pipelines

Artifact Repository Browser | Set Me Up | Deploy | Actions

SET ME UP

Tool: Conan

Repository: myconanrepo

Type password to insert your credentials to the code snippets

Type Password

12. Copy the URL

General

For your Conan command line client to work with this Conan repository, you first need to add the repository to your client configuration using the following command:

```
1  conan remote add <REMOTE> http://34.67.115.149:8081/artifactory/api/conan/myconanrepo
```

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

Trash Can

Copyright @ 2020 JFrog - All rights reserved.

Exercise 9 - Upload Packages to Artifactory

3:00

```
$ conan remote add artifactory <URL from SetMeUp>
$ conan remote list
$ conan upload "hello*" -r artifactory --all

# search remote repositories
$ conan search "*" -r=artifactory
$ conan search hello/0.1@user/testing -r=artifactory
# Navigate to Artifactory WebUI and check!
```

Copyright @ 2020 JFrog - All rights reserved.

Exercise 9 - 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
# Can re-add with: $ conan remote add conan-center https://conan.bintray.com
```

$ ../catchup.sh # option 9

Copyright @ 2020 JFrog - All rights reserved.

Exercise 10 - Exploring the Conan Cache

Goal:

- Learn a little about our Cache

Task:

- Clear our Conan cache and explore it

Success:

- If you can find your Conan cache on your filesystem

Copyright @ 2020 JFrog - All rights reserved.

Exercise 10 - Exploring the Conan Cache

```
$ ls ~/.conan # look inside your conan home
$ ls ~/.conan/data # look inside your conan cache
$ ls ~/.conan/data/hello/0.1/user/testing
$ ls ~/.conan/data/hello/0.1/user/testing/package/<id>/include
$ ls ~/.conan/data/hello/0.1/user/testing/package/<id>/lib

$ conan remove "*" –f
$ conan search # local cache should be empty
$ ls ~/.conan/data
```

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

Copyright @ 2020 JFrog - All rights reserved.

Exercise 11 - Consume packages from Artifactory

Goal:

- Consume packages from Artifactory

Task:

- Build the timer application using ONLY our “artifactory” remote

Success:

- Timer builds and runs, and we did not interact with ConanCenter

Copyright @ 2020 JFrog - All rights reserved.

Exercise 11 - Consume packages from Artifactory

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

$ ../catchup.sh # option 11

Copyright @ 2020 JFrog - All rights reserved.

Outline

- Introduction
- Consume Conan packages
- Create Conan packages
- Uploading packages to Artifactory
- **Build configuration & cross-build**

Copyright @ 2020 JFrog - All rights reserved.

Exercise 12 - Using options for shared/static

Goal:

- Learn about options vs. settings for package configuration

Task:

- Create and test (with test_package) packages for “hello” library with shared libraries

Success:

- “Hello world” messages are printed on screen.

Copyright @ 2020 JFrog - All rights reserved.

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 (e.g. shared, static, active FPU, etc)

Copyright @ 2020 JFrog - All rights reserved.

Exercise 12 – 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
$ conan search hello/0.1@user/testing
$ ../catchup.sh # option 12
```

Copyright @ 2020 JFrog - All rights reserved.

Exercise 12 - Shared libraries runtime location?

create_sources/test_package/conanfile.py

```
class HelloTestConan(ConanFile):
    settings = "os", "compiler", "build_type", "arch"
    generators = "cmake"

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

    def imports(self):
        self.copy('*.so*', dst='bin', src='lib')

    def test(self):
        ...
```

Copyright @ 2020 JFrog - All rights reserved.

Exercise 13 - Define and use custom options

Goal:

- Learn to define custom options in our package recipes

Task:

- Create and test packages for the “greet” library that output messages in different languages based on an input option

Success:

- “Hello world” messages in different languages are printed on screen.

Copyright @ 2020 JFrog - All rights reserved.

Exercise 13 – Custom option “language”

$ cd training/create_options

src/hello.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_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})
```

Copyright @ 2020 JFrog - All rights reserved.

Exercise 13 – Custom option “language”

$ cd training/create_options

conanfile.py

```
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()
```

Copyright @ 2020 JFrog - All rights reserved.

Exercise 13 – Custom option “language”

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

$ ../catchup.sh # option 13
```

Copyright @ 2020 JFrog - All rights reserved.

Exercise 14 – Configuration values and errors

Goal:

- Learn about configuration error messages and defining custom settings
- Learn about new commands (inspect, get)

Task:

- Introduce non-existing options and settings in the commands

Success:

- See different error messages for different configuration inputs

Copyright @ 2020 JFrog - All rights reserved.

Exercise 14 – Configuration values and errors

```
$ conan create . user/testing -o greet:language=Italian  # Error
# and to check available options?
$ conan inspect greet/0.1@user/testing
$ conan inspect zlib/1.2.11@
$ conan get zlib/1.2.11@

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

Copyright @ 2020 JFrog - All rights reserved.

Exercise 14 – 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"]
```

Copyright @ 2020 JFrog - All rights reserved.

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"]
```

Copyright @ 2020 JFrog - All rights reserved.

Exercise 15 - Profiles and cross-compiling

Goal:

- Learn about profile files to manage configuration definitions
- Cross compile the hello package to Raspberry PI (arm architecture)

Task:

- Use input profiles to create a “hello” package that can be used in a RPI

Success:

- “hello” package is created in the local cache with configuration for a RPI

Copyright @ 2020 JFrog - All rights reserved.

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

Copyright @ 2020 JFrog - All rights reserved.

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++
```

Copyright @ 2020 JFrog - All rights reserved.

Conan profiles

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

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

Copyright @ 2020 JFrog - All rights reserved.

Exercise 15 – 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++
```

Copyright @ 2020 JFrog - All rights reserved.

Exercise 15 – 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 15
```

Copyright @ 2020 JFrog - All rights reserved.

Profiles: including and variables

```
CROSS_GCC=arm-linux-gnueabihf

include(default)

[settings]
arch=armv7

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

Copyright @ 2020 JFrog - All rights reserved.

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
```

Copyright @ 2020 JFrog - All rights reserved.

Profiles: composition

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

Copyright @ 2020 JFrog - All rights reserved.

Homework

- Package the open source library Pico JSON: https://github.com/kazuho/picojson.git
- Tips:
  - It is a header-only library, doesn't need **settings** or **build()** method
  - Hint: Use "conan new --help"
  - Use "test_package" functionality, get some code from their github.
- Upload it to your Artifactory
- Use it in the "consumer" project.

Copyright @ 2020 JFrog - All rights reserved.

General Notes / Tips

1. Read the output carefully
2. Know your local Conan cache
   a. This will help with debugging, especially during conan create
   b. Default: ~/.conan/data
3. Follow standard names and conventions
   a. settings and options
4. ConanCenter does not use user/channel
   a. Custom packages you create should use user/channel

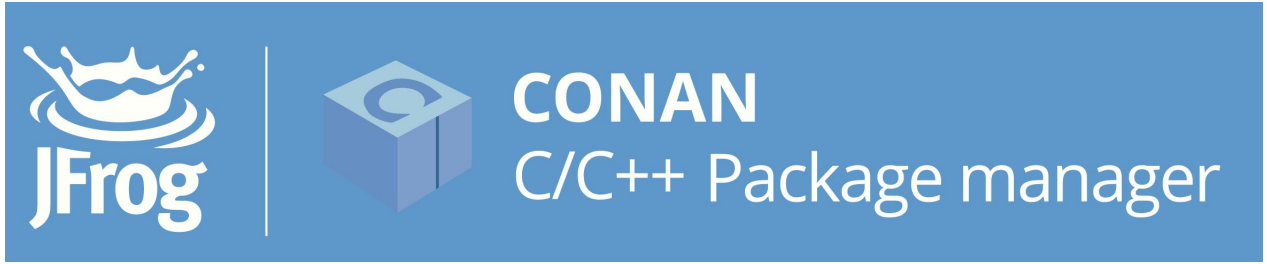
Copyright @ 2020 JFrog - All rights reserved.

Resources

- Docs: https://docs.conan.io/
  - Read carefully, explore.
- Issues:
  - CppLang slack (community)
  - Github issues (https://github.com/conan-io/conan) “official” support
- Following trainings:
  - Advanced, CI/CD for registered attendees: conandays@jfrog.com
- Want some help from Verifa?
  - info@verifa.io | Contact Jacob in Slack

Copyright @ 2020 JFrog - All rights reserved.

THANK YOU!



Copyright @ 2020 JFrog - All rights reserved.