С Conan за кроссплатформенностью

Семен Буденков

IntelliVision

C++ Russia 2023

IntelliVision
AI and Video Analytics for Smart Cameras

semen.budenkov@intelli-vision.com

Содержание

- Управление зависимостями в C++ проектах.
- Путь от Makefile и Visual Studio проектов к CMake и Conan.
- Кроссплатформенная разработка с Conan.

Опрос

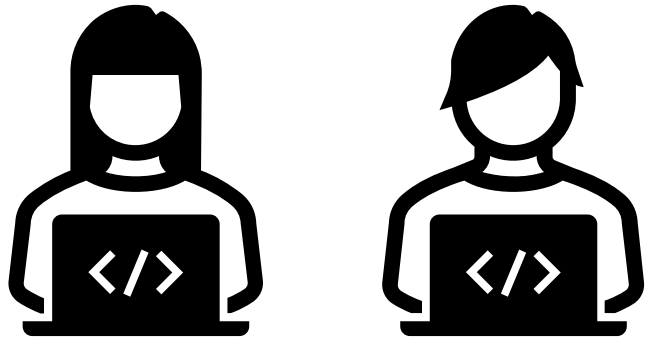
- Вы пользуетесь менеджером пакетов для C++?
  - Да
  - Нет
- Каким менеджером вы пользовались?
  - Conan
  - vcpkg
  - Другой
- Если да, то использовали ли вы больше одного менеджера одновременно?
  - Да
  - Нет
- Как много триплетов (CPU архитектура, компилятор и пр.) приходится поддерживать?
  - 1
  - 1 - 5
  - 5 - 10
  - > 10

Управление зависимостями в C++ проектах

Вместо вступления

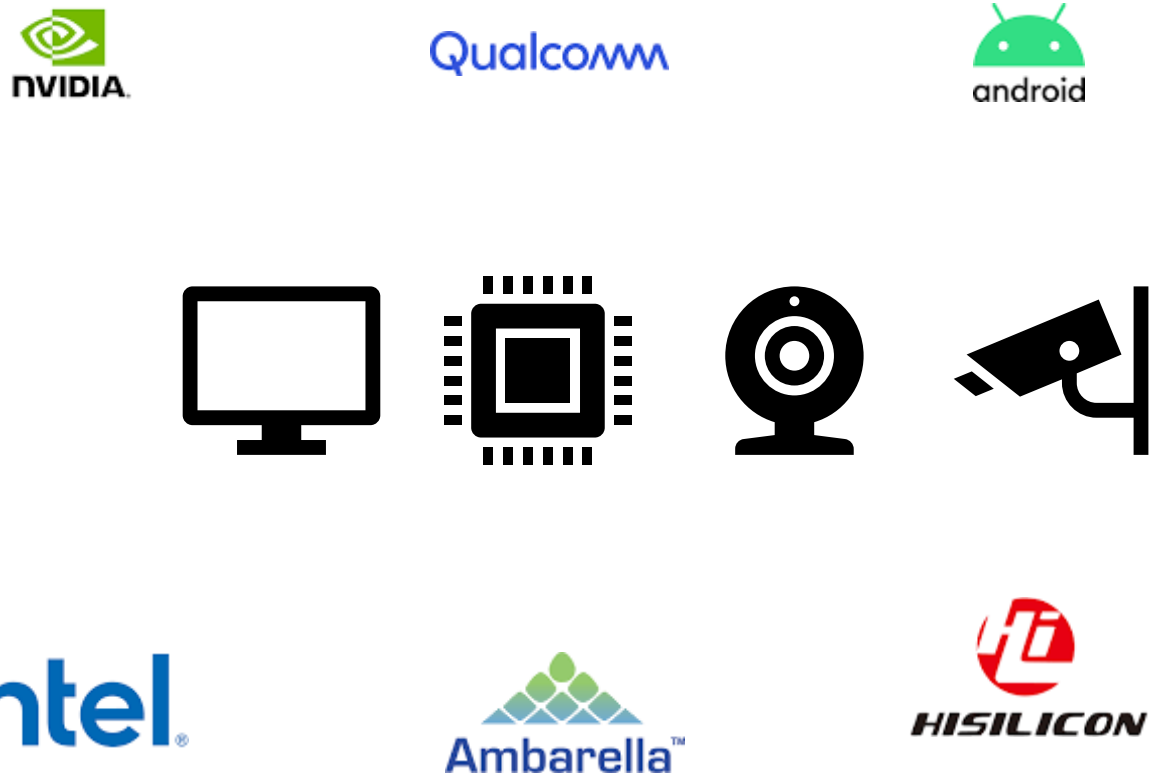
Инфраструктура разработки

- Около 10 активных проектов.
- Продукт – библиотека CV (computer vision) аналитики с C API.
- У каждого продукта ~5-10 зависимостей.
- Число активных разработчиков ~10-20.
- Число инженеров по поддержке разработки – 1.



Список платформ

- Intel CPU
- NVIDIA GPU
- NVIDIA Jetson
- NPU (Intel, Hailo)
- ARM (Ambarella, HiSilicon, Qualcomm)
- Android устройства



Сложности C++ разработки

- Q6 Which of these do you find frustrating about C++ development?
  - Управление зависимостями проектов.
  - Время построения.
  - Настройка построения и расширение CI.
  - Развертывание окружения.
  - Отладка кода и поиск ошибок.

	MAJOR PAIN POINT	MINOR PAIN POINT	NOT A SIGNIFICANT ISSUE FOR ME	TOTAL	WEIGHTED AVERAGE
Managing libraries my application depends on	47.37% 810	35.09% 600	17.54% 300	1,710	2.30
Build times	43.34% 735	37.56% 637	19.10% 324	1,696	2.24
Setting up a continuous integration pipeline from scratch (automated builds, tests, ...)	31.35% 531	40.85% 692	27.80% 471	1,694	2.04
Managing CMake projects	31.24% 527	36.10% 609	32.66% 551	1,687	1.99
Concurrency safety: Races, deadlocks, performance bottlenecks	28.17% 480	41.37% 705	30.46% 519	1,704	1.98
Setting up a development environment from scratch (compiler, build system, IDE, ...)	26.83% 459	41.09% 703	32.09% 549	1,711	1.95
Debugging issues in my code	18.52% 313	51.24% 866	30.24% 511	1,690	1.88
Parallelism support: Using more CPU/GPU/other cores to compute an answer faster	23.24% 393	36.84% 623	39.92% 675	1,691	1.83
Memory safety: Bounds safety issues (read/write beyond the bounds of an object or array)	18.92% 323	37.55% 641	43.53% 743	1,707	1.75
Memory safety: Use-after-delete/free (dangling pointers, iterators, spans, ...)	18.83% 321	34.13% 582	47.04% 802	1,705	1.72
Managing Makefiles	20.21% 333	21.48% 354	58.31% 961	1,648	1.62
Security issues: Overlaps with “safety” but includes other issues (secret disclosure, vulnerabilities, exploits, ...)	11.42% 193	34.44% 582	54.14% 915	1,690	1.57
Managing MSBuild projects	16.77% 274	20.99% 343	62.24% 1,017	1,634	1.55
Type safety: Using an object as the wrong type (unsafe downcasts, unsafe unions, ...)	10.90% 186	32.69% 558	56.41% 963	1,707	1.54
Memory safety: Forgot to delete/free (memory leaks)	11.60% 198	28.24% 482	60.16% 1,027	1,707	1.51
Moving existing code to the latest language standard	7.92% 135	26.88% 458	65.20% 1,111	1,704	1.43

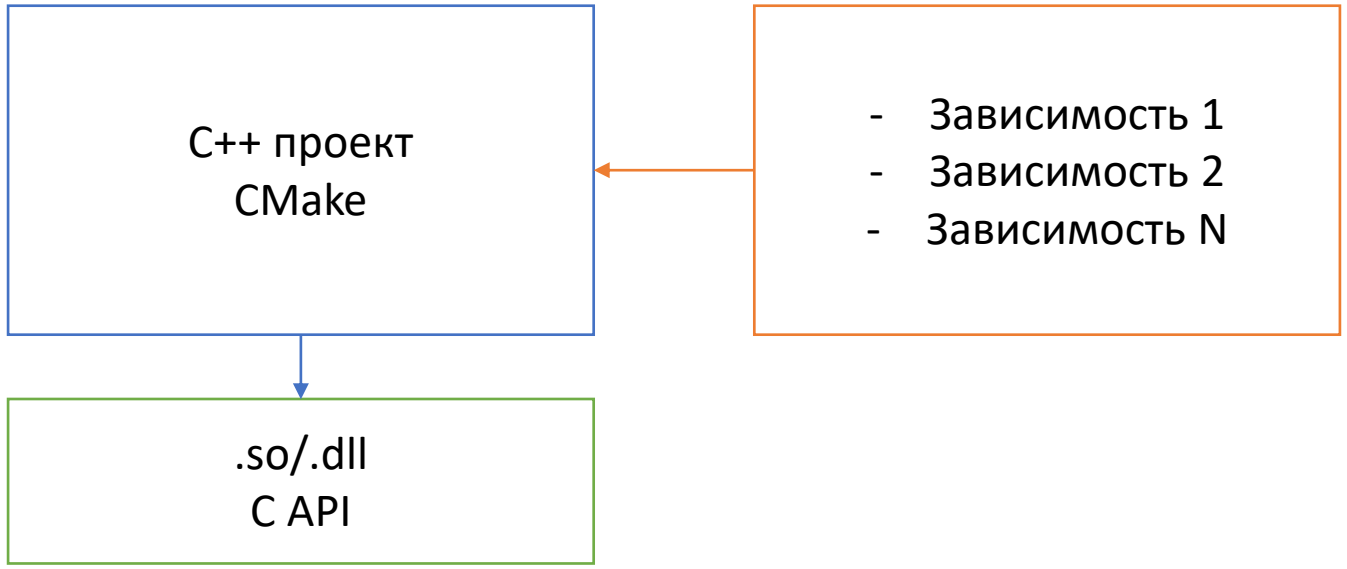
2023 Annual C++ Developer Survey “Lite”
https://isocpp.org/blog/2023/04/results-summary-2023-annual-cpp-developer-survey-lite

Управление зависимостями

- Сборка библиотек и использование сетевого хранилища.
- Исходный код внутри проекта.
- Системные библиотеки и менеджеры:
  - apt
  - yum
  - pacman
- Менеджеры зависимостей:
  - Conan
  - vcpkg

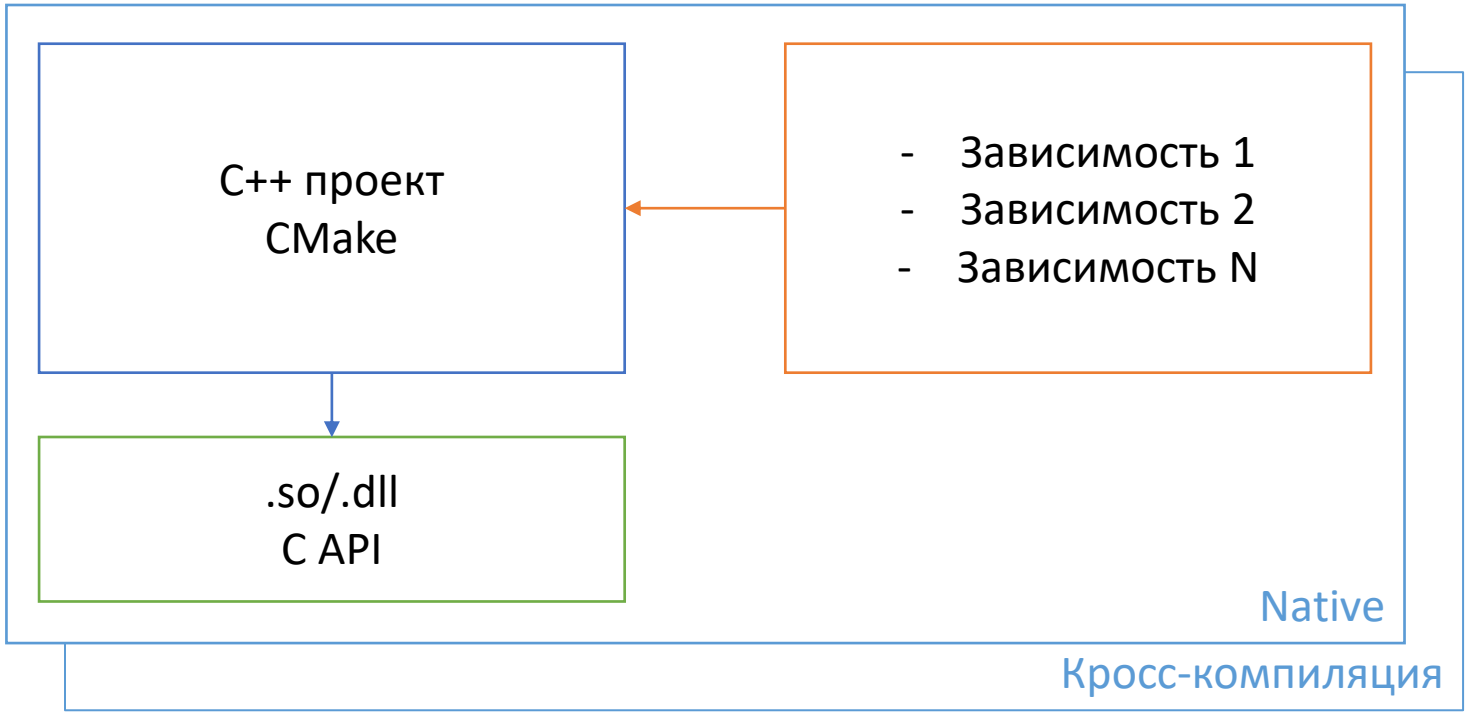
ANSWER CHOICES	RESPONSES	
The library source code is part of my build	68.11%	1,162
I compile the libraries separately using their instructions	48.30%	824
System package managers (e.g., apt, brew, ...)	36.23%	618
I download prebuilt libraries from the Internet	27.43%	468
Vcpkg	21.34%	364
Conan	18.93%	323
Other (please specify)	11.25%	192
Nuget	7.50%	128
None of the above, I do not have any dependencies	1.41%	24
Total Respondents: 1,706		

Типовой проект



- Добавление новой библиотеки/версии в зависимости.
- Сборка под новые устройства.

Типовой кроссплатформенный проект



- Native сборка на устройстве.
- Использование серверов для построения.

Используемое решение

- CMake – кроссплатформенное средство автоматизации сборки.
- Менеджер пакетов (Conan, vcpkg) для управления внешними зависимостями.
- Плоская структура хранения подпроектов.
- Конечный продукт и зависимости «смотрят» на одну версию.

Путь от Makefile и Visual Studio проектов к CMake + Conan

Горький опыт?

Было

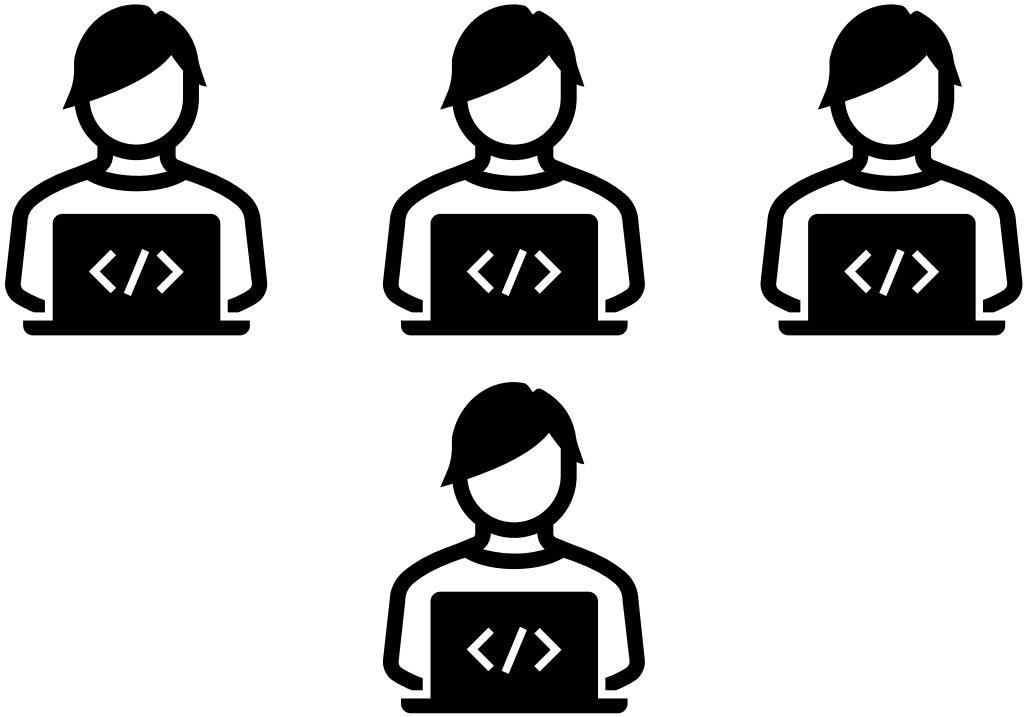
- Размер артефактория с зависимостями:
  - Дисковое пространство: **482.3 Гб**
  - Число библиотек зависимостей: ~50 шт.
- Размер репозитория FFmpeg:
  - Дисковое пространство: **4.4 Гб**
  - Число конфигураций: 80 шт.
- Время построения:
  - Время сборки проекта: **90 сек**
  - Добавление новой версии или зависимости: 1-3 дня

Стало

- Размер кеша с зависимостями:
  - Дисковое пространство: **10 Гб**
  - Число библиотек зависимостей: 84 шт.
- Размер репозитория FFmpeg :
  - Дисковое пространство: **900 Мб**
  - Число конфигураций: 2 шт.
- Время построения:
  - Время сборки проекта: **45 сек**
  - Добавление новой версии или зависимости: 1 день

Время на внедрение

- **1 этап:** внедрение
  - Планировалось: 6 месяцев
  - По факту: ~9 месяцев
  - Люди: 1 человек (1-3 временно)
- **2 этап:** перевод проектов
  - Первый проект: 1 месяц
  - Последующие: 1-2 недели
  - Люди: 1 человек



https://decovar.dev/blog/2022/02/06/cpp-dependencies-with-conan/

Кроссплатформенная разработка с Conan

А какие варианты?

IntelliVision
AI and Video Analytics for Smart Cameras

Список поддерживаемых конфигураций 1

- aarch64-linux-android-4.9.x-x64-debug
- aarch64-linux-android-4.9.x-x64-release
- aarch64-linux-gnu-10.2.1-x64-debug
- aarch64-linux-gnu-10.2.1-x64-release
- aarch64-linux-gnu-5.4.0-x64-debug
- aarch64-linux-gnu-5.4.0-x64-release
- aarch64-linux-gnu-6.3.1-x64-debug
- aarch64-linux-gnu-6.3.1-x64-release
- aarch64-linux-gnu-7-x64-debug
- aarch64-linux-gnu-7-x64-release
- aarch64-linux-gnu-7.1.1-x64-debug
- aarch64-linux-gnu-7.1.1-x64-release
- aarch64-linux-gnu-7.3.1-x64-debug
- aarch64-linux-gnu-7.3.1-x64-release
- aarch64-linux-gnu-8.2.1-x64-debug
- aarch64-linux-gnu-8.2.1-x64-release
- aarch64-linux-gnu-9-x64-debug
- aarch64-linux-gnu-9-x64-release
- aarch64-none-linux-android21-debug
- aarch64-none-linux-android21-release
- aarch64-oe-linux-8.2.0-x64-debug
- aarch64-oe-linux-8.2.0-x64-release
- aarch64-poky-linux-7.3.0-x64-debug
- aarch64-poky-linux-7.3.0-x64-release
- aarch64-poky-linux-9.2.0-x64-debug
- aarch64-poky-linux-9.2.0-x64-release
- arm-axis-linux-gnueabi-4.3.1-x32-debug
- arm-axis-linux-gnueabi-4.3.1-x32-release
- arm-axis-linux-gnueabihf-4.7.2-x32-debug

Список поддерживаемых конфигураций 2

- arm-buildroot-linux-uclibcgnueabihf-4.9.4-x32-debug
- arm-buildroot-linux-uclibcgnueabihf-4.9.4-x32-release
- arm-hisiv300-linux-uclibcgnueabi-4.8.3-x32-debug
- arm-hisiv300-linux-uclibcgnueabi-4.8.3-x32-release
- arm-hisiv400-linux-gnueabi-4.8.3-x32-debug
- arm-hisiv400-linux-gnueabi-4.8.3-x32-release
- arm-hisiv500-linux-uclibcgnueabi-4.9.4-x32-debug
- arm-hisiv500-linux-uclibcgnueabi-4.9.4-x32-release
- arm-hisiv600-linux-gnueabi-4.9.4-x32-debug
- arm-hisiv600-linux-gnueabi-4.9.4-x32-release
- arm-linux-androideabi-4.9.x-x32-debug
- arm-linux-androideabi-4.9.x-x32-release
- arm-linux-gnueabi-6.3.0-x32-debug
- arm-linux-gnueabi-6.3.0-x32-release
- arm-linux-gnueabihf-4.8.2-x32-debug
- arm-linux-gnueabihf-4.8.2-x32-release
- arm-linux-gnueabihf-4.9.1-x32-debug
- arm-linux-gnueabihf-4.9.1-x32-release
- arm-linux-gnueabihf-5.2.1-x32-debug
- arm-linux-gnueabihf-5.2.1-x32-release
- arm-linux-gnueabihf-5.4.0-x32-debug
- arm-linux-gnueabihf-5.4.0-x32-release
- arm-linux-gnueabihf-7.1.1-x32-debug
- arm-linux-gnueabihf-7.1.1-x32-release
- arm-linux-gnueabihf-8-x32-debug
- arm-linux-gnueabihf-8-x32-release
- arm-linux-gnueabihf-8.2.1-x32-debug
- arm-linux-gnueabihf-8.2.1-x32-release
- arm-linux-gnueabihf-8.3.0-x32-debug
- arm-linux-gnueabihf-9.1.0-x32-debug
- arm-linux-gnueabihf-9.1.0-x32-release

Список поддерживаемых конфигураций 3

- arm-none-linux-android21-debug
- arm-none-linux-android21-release
- arm-none-linux-gnueabi-4.3.3-x32-debug
- arm-none-linux-gnueabi-4.3.3-x32-release
- arm-none-linux-gnueabi-4.6.1-x32-debug
- arm-none-linux-gnueabi-4.6.1-x32-release
- arm-oe-linux-gnueabi-4.9.3-x32-debug
- arm-oe-linux-gnueabi-4.9.3-x32-release
- arm-openwrt-linux-gnueabi-7.3.0-x32-debug
- arm-openwrt-linux-gnueabi-7.3.0-x32-release
- arm-ov798-linux-uclibcgnueabi-4.8.4-x32-debug
- arm-ov798-linux-uclibcgnueabi-4.8.4-x32-release
- arm-poky-linux-gnueabi-4.9.1-x32-debug
- arm-poky-linux-gnueabi-4.9.1-x32-release
- arm-poky-linux-gnueabi-6.2.0-x32-debug
- arm-poky-linux-gnueabi-6.2.0-x32-release
- arm-poky-linux-gnueabi-9.2.0-x32-debug
- arm-poky-linux-gnueabi-9.2.0-x32-release
- arm-unknown-linux-uclibcgnueabi-4.4.0-x32-debug
- arm-unknown-linux-uclibcgnueabi-4.4.0-x32-release
- i686-tattile-linux-gnu-5.3.0-x32-debug
- i686-tattile-linux-gnu-5.3.0-x32-release
- i686-verint730-linux-gnu-7.3.0-x32-debug
- i686-verint730-linux-gnu-7.3.0-x32-release
- win-x64-v140-Release
- win-x86-v140-Release

Список поддерживаемых конфигураций 4

- x86_64-linux-gnu-11-x64-debug
- x86_64-linux-gnu-11-x64-release
- x86_64-linux-gnu-17.0.2-x64-debug
- x86_64-linux-gnu-17.0.2-x64-release
- x86_64-linux-gnu-4.8-x32-debug
- x86_64-linux-gnu-4.8-x32-release
- x86_64-linux-gnu-4.8-x64-debug
- x86_64-linux-gnu-4.8-x64-release
- x86_64-linux-gnu-5.4.0-x32-debug
- x86_64-linux-gnu-5.4.0-x32-release
- x86_64-linux-gnu-5.4.0-x64-debug
- x86_64-linux-gnu-5.4.0-x64-release
- x86_64-linux-gnu-6.3.0-x64-debug
- x86_64-linux-gnu-6.3.0-x64-release
- x86_64-linux-gnu-7-x32-debug
- x86_64-linux-gnu-7-x32-release
- x86_64-linux-gnu-7-x64-debug
- x86_64-linux-gnu-7-x64-release
- x86_64-linux-gnu-7-x64-release-ssl
- x86_64-linux-gnu-9-x64-debug
- x86_64-linux-gnu-9-x64-release
- x86_64-redhat-linux-4.4.7-x64-debug
- x86_64-redhat-linux-4.4.7-x64-release
- x86_64-redhat-linux-5.3.1-x64-debug
- x86_64-redhat-linux-5.3.1-x64-release
- x86_64-redhat-linux-8-x64-debug
- x86_64-redhat-linux-8-x64-release

GNU триплеты в Conan

- **Build**: Платформа на которой будут запущены инструменты сборки.
- **Host**: Платформа на которой будет запущен собранный тулчейн.
- **Target**: Определяет платформу для которой генерируются файлы.

Build == Host Host == Target	Build == Host Host != Target

https://docs.conan.io/1/systems_cross_building/cross_building.html
https://gcc.gnu.org/onlinedocs/gccint/Configure-Terms.html

Кросс-платформенная сборка с Conan

- profile
  - [settings]: os, arch, compiler, build_type
  - [env]: PATH, CXX, CONAN_CMAKE_SYSROOT
  - [conf]: tools.build:sysroot
- tool_requires
  - Тулчейн упакован в пакет Conan.
  - Профиль **Host** содержит секцию [tool_requires] с тулчейном.
  - Рецепт содержит секцию tool_requires с тулчейном.

```
toolchain_path=$toolchain_root_dir/linaro-aarch64-2020.09-gcc10.2-linux5.4
toolchain_prefix=aarch64-linux-gnu

[settings]
os               = Linux
arch             = armv8
compiler         = gcc
compiler.version = 10
compiler.libcxx  = libstdc++11
build_type       = Release

[env]
CONAN_CMAKE_TOOLCHAIN_FILE = ${CMAKE_TOOLCHAIN_FILES}/aarch64-linaro-gcc10.2-linux.cmake
PATH=[$toolchain_path/aarch64-linux-gnu/bin]
CHOST=aarch64-linux-gnu
CC=$toolchain_path/bin/$toolchain_prefix-gcc
CXX=$toolchain_path/bin/$toolchain_prefix-g++
STRIP=$toolchain_path/bin/$toolchain_prefix-strip
```

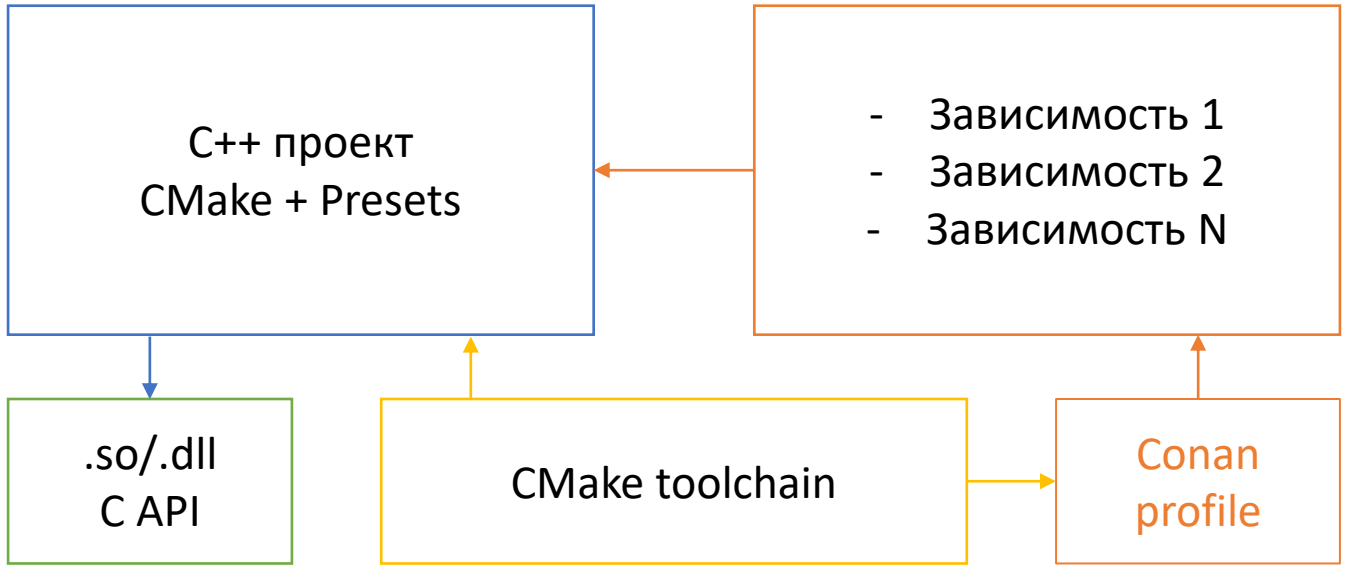
```
[tool_requires]
my_toolchain/0.1@user/channel
```

```
class MyPkg(ConanFile):
    tool_requires = "my_toolchain/0.1@user/channel"

    def build_requirements(self):
        if platform.system() == "Windows":

self.tool_requires("my_toolchain/0.1@user/channel")
```

```
import os
from conans import ConanFile

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

    # Implement source() and build() as usual

    def package(self):
        # Copy all the required files for your toolchain
        self.copy("*", dst="", src="toolchain")

    def package_info(self):
        bin_folder = os.path.join(self.package_folder, "bin")
        self.env_info.CC = os.path.join(bin_folder, "mycompiler-cc")
        self.env_info.CXX = os.path.join(bin_folder, "mycompiler-cxx")
        self.env_info.SYSROOT = self.package_folder
```

https://docs.conan.io/1/systems_cross_building/cross_building.html

Типовой кроссплатформенный проект




```
CMAKE_SYSTEM_NAME Linux
CMAKE_SYSTEM_PROCESSOR aarch64
CMAKE_C_COMPILER aarch64-linux-gnu-gcc
CMAKE_CXX_COMPILER aarch64-linux-gnu-g++
```

```
[settings]
os                = Linux
arch              = armv8
compiler          = gcc
compiler.version  = 7
compiler.libcxx   = libstdc++11
build_type        = Release
```

https://cmake.org/cmake/help/book/mastering-cmake/chapter/Cross%20Compiling%20With%20CMake.html

CMake preset jetson.json

```
{
    "version": 4,
    "cmakeMinimumRequired":
    {
        "major": 3,
        "minor": 23,
        "patch": 0
    },
    "configurePresets":
    [
        {
            "name": "jetson-base",
            "description": "Configuration for NVIDIA Jetson platform",
            "hidden": true,
            "toolchainFile": "${sourceDir}/cmake_common/aarch64-gccX-linux.cmake",
            "cacheVariables":
            {
                "CONAN_PROFILE": "nvidia-jetson"
            }
        }
    ]
}
```

CMake toolchain aarch64-gccX-linux.cmake

```
set(CMAKE_SYSTEM_NAME Linux)
set(CMAKE_SYSTEM_VERSION 1)
set(CMAKE_SYSTEM_PROCESSOR aarch64)

set(CMAKE_C_COMPILER aarch64-linux-gnu-gcc)
set(CMAKE_CXX_COMPILER aarch64-linux-gnu-g++)
set(CMAKE_CXX_FLAGS "-Wall")
set(CMAKE_C_FLAGS "-Wall")

# WARNING: Specifying CMAKE_FIND_ROOT_PATH breaks build OpenCV with
conan
#set(CMAKE_FIND_ROOT_PATH /usr/aarch64-linux-gnu)

set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
```

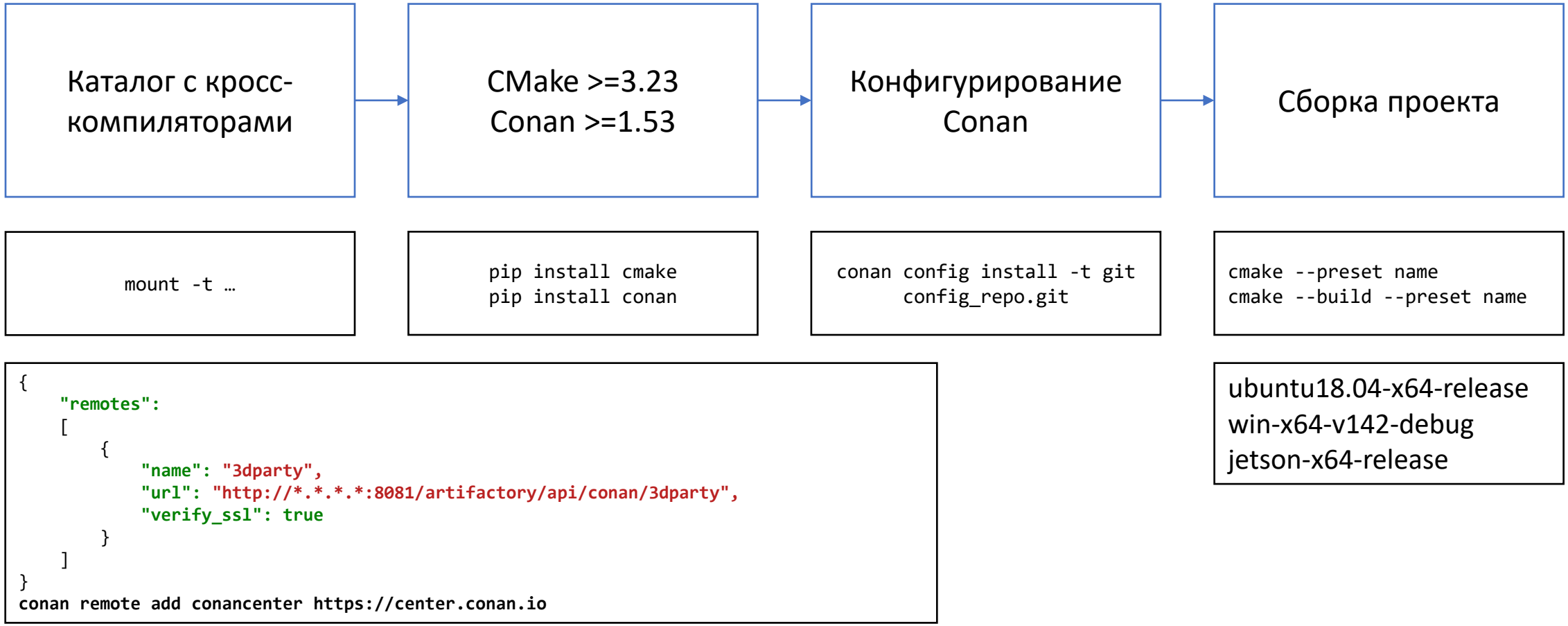
Conan profile nvidia-jetson

```
include(toolchains/aarch64-none-gcc7-linux)
include(packages/packages_common)
[settings]
os.platform = Nvidia_Jetson
[options]
opencv:with_gtk=False
```

```
include(boost_common)
include(opencv_common)
[settings]
ffmpeg:build_type=Release
opencv:build_type=Release
```

```
target_host=aarch64-linux-gnu
[settings]
os                  = Linux
arch                = armv8
compiler            = gcc
compiler.version    = 7
compiler.libcxx     = libstdc++11
build_type          = Release
[env]
CONAN_CMAKE_TOOLCHAIN_FILE =
${CMAKE_TOOLCHAIN_FILES}/aarch64-
gccX-linux.cmake
PATH=[/usr/$target_host/bin]
CC=/usr/bin/$target_host-gcc
CXX=/usr/bin/$target_host-g++
```

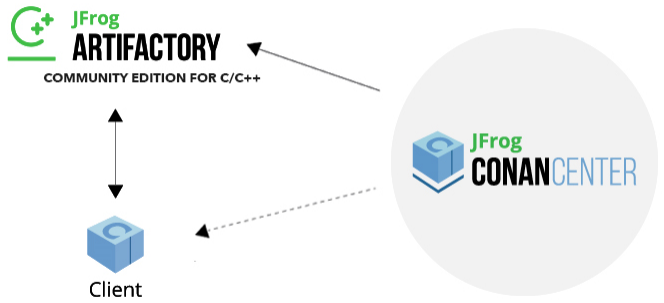
Настройка окружения

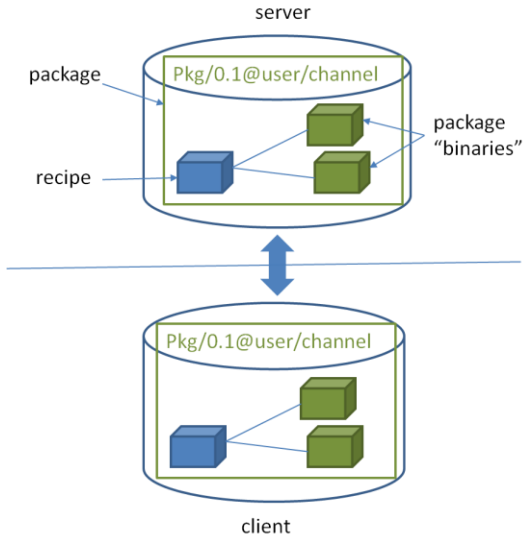


Установка пакетов зависимостей

```
list(APPEND CONAN_DEPS boost/1.79.0@conan/stable rapidjson/cci.20211112@conan/stable  spdlog/1.10.0@conan/stable)
conan_cmake_configure(REQUIRES ${CONAN_DEPS} GENERATORS CMakeDeps cmake OPTIONS ${CONAN_OPTIONS})
conan_cmake_autodetect(settings)
```

```
conan_cmake_install(
    PATH_OR_REFERENCE
    .
    ${CONAN_INSTALL_OPTION}
    PROFILE
    ${CONAN_PROFILE}
    PROFILE_BUILD
    ${CONAN_PROFILE_BUILD}
    GENERATOR
    virtualrunenv
    SETTINGS
    ${settings})
```






```
set(CONAN_INSTALL_OPTION UPDATE CACHE STRING
    "Check the remote for a newer version and/or revision of the dependencies")
```

https://github.com/conan-io/cmake-conan
https://docs.conan.io/1/howtos/vs2017_cmake.html#using-cmake-conan

Пакетирование

- Внутренние рецепты (~20):
  - Qualcomm: SNPE
  - NVIDIA: CUDA, cuDNN, TensorRT
  - Intel: OpenVINO, TBB
  - Google: TensorFlow Lite, XNNPACK
  - Hailo.AI
- Рецепты Conan (> 80):
  - Boost
  - FFmpeg
  - OpenCV

Создание пакетов для SDK

```
from conans import ConanFile, tools
class MySDKConanFile(ConanFile):
    name="mysdk"
    ...
    def build(self):
        binary_info = self.conan_data["sources"][self.version][suffix]
        tools.get(**binary_info, strip_root=True)
    def package(self):
        if self.settings.os == "Windows":
            self.copy("*", dst="bin", src="bin")
            self.copy("*", dst="lib", src="lib", symlinks=True)
            self.copy("*", dst="include", src="include")
    def package_info(self):
        self.cpp_info.sharedlinkflags = [f"L{self.package_folder}/lib_folder"]
```

```
<package_id>
|- include
|- lib
|    |- lib_in_sdk.so
| - conaninfo.txt
| - conanmanifest.txt
```

```
package
|- all
|    |- conandata.yml
|    |- conanfile.py
|- config.yml
```

```
target_link_libraries(my_targe
    PUBLIC
    mysdk::mysdk
    lib_in_sdk
)
```

https://docs.conan.io/2/tutorial/creating_packages/other_types_of_packages/package_prebuilt_binaries.html

Conan Host профиль

- Основные секции:
  - [settings]
  - [options]
- Формирование package_id:
  - [settings], [options], [requirements]
  - <userhome>/.conan/settings.yml

```
[settings]
os                = Linux
arch              = armv7
compiler          = gcc
compiler.version  = 4.9
compiler.libcxx   = libstdc++
build_type        = Release
os.platform=Qualcomm
os.platform.model=MSM8953

[options]
opencv*:with_gtk = False
...
```

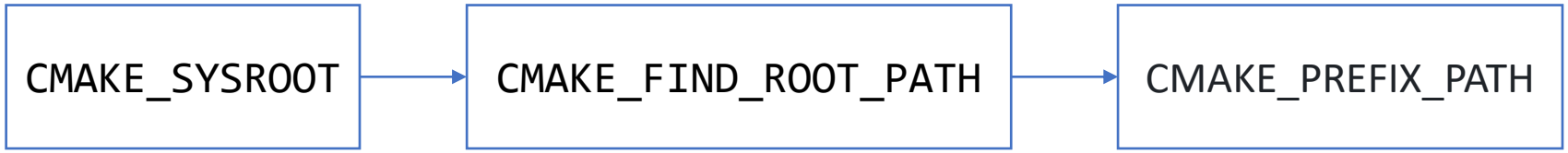
```
Linux:
    platform:
        Qualcomm:
            model: [None, APQ8053, QCS610, MSM8953]
```

https://blog.conan.io/2019/09/27/package-id-modes.html
https://docs.conan.io/1/creating_packages/define_abi_compatibility.html#versioning-schema

Cmake/GNU Autotools/Conan

```
CMAKE_TOOLCHAIN_FILE
CMAKE_SYSTEM_NAME
CMAKE_SYSTEM_PROCESSOR
CMAKE_SYSROOT
CMAKE_FIND_ROOT_PATH
CMAKE_FIND_ROOT_PATH_MODE_PROGRAM
CMAKE_FIND_ROOT_PATH_MODE_LIBRARY
CMAKE_FIND_ROOT_PATH_MODE_INCLUDE
```

```
PATH
CHOST
CC
CXX
AR
AS
CFLAGS
CXXFLAGS
```



https://docs.conan.io/1/reference/env_vars.html#cmake-related-variables
https://discourse.cmake.org/t/cmake-find-root-path-and-toolchain-files/3294/3

Conan Host профиль

- Дополнительные секции:
  - [conf]
  - [env]/[buildenv]

```
[conf]
tools.build:sysroot=$sysroot
tools.cmake.cmaketoolchain:user_toolchain=["$cmake_toolchain_file"]

[env]
# [env] is deprecated! Use [buildenv] instead
PATH=[$iv_toolchain_root_dir/poky/oecore-x86_64/sysroots/x86_64-
oesdk-linux/usr/libexec/arm-oe-linux-gnueabi/gcc/arm-oe-linux-
gnueabi/4.9.3]
CONAN_CMAKE_SYSTEM_PROCESSOR=armv7
CONAN_CMAKE_FIND_ROOT_PATH_MODE_PROGRAM=NEVER
CONAN_CMAKE_FIND_ROOT_PATH_MODE_LIBRARY=ONLY
CONAN_CMAKE_FIND_ROOT_PATH_MODE_INCLUDE=ONLY

CC="$toolchain_path/$toolchain_prefix-gcc --sysroot=$sysroot"
CXX="$toolchain_path/$toolchain_prefix-g++ --sysroot=$sysroot"
AS="$toolchain_path/$toolchain_prefix-as"
CXXFLAGS="$cc_cxx_flags_str"
CFLAGS="$cc_cxx_flags_str -std=gnu99"
```

Conan Host профиль

- CMake:
  - CMake
  - CMakeToolchain
  - CMakeDeps
- GNU:
  - Autotools
  - AutotoolsToolchain
  - AutotoolsDeps

from conan.tools.cmake import CMake

from conans import CMake

from conans import AutoToolsBuildEnvironment

```
[conf]
tools.build:sysroot=$sysroot
tools.cmake.cmaketoolchain:user_toolchain=["$cmake_toolchain_file"]

[env]
# [env] is deprecated! Use [buildenv] instead
PATH=[$iv_toolchain_root_dir/poky/oecore-x86_64/sysroots/x86_64-
oesdk-linux/usr/libexec/arm-oe-linux-gnueabi/gcc/arm-oe-linux-
gnueabi/4.9.3]
CONAN_CMAKE_SYSTEM_PROCESSOR=armv7
CONAN_CMAKE_FIND_ROOT_PATH_MODE_PROGRAM=NEVER
CONAN_CMAKE_FIND_ROOT_PATH_MODE_LIBRARY=ONLY
CONAN_CMAKE_FIND_ROOT_PATH_MODE_INCLUDE=ONLY

CC="$toolchain_path/$toolchain_prefix-gcc --sysroot=$sysroot"
CXX="$toolchain_path/$toolchain_prefix-g++ --sysroot=$sysroot"
AS="$toolchain_path/$toolchain_prefix-as"
CXXFLAGS="$cc_cxx_flags_str"
CFLAGS="$cc_cxx_flags_str -std=gnu99"
```

https://docs.conan.io/1/reference/conanfile/tools/cmake/cmaketoolchain.html#cross-building

Conan Host профиль

```python
import os
from conans import ConanFile, CMake, MSBuild, to

class SimdConan(ConanFile):
    ...

    def _configure_cmake(self):
        cmake = CMake(self)
        is_x86 = self.settings.arch == "x86" or
        cmake.definitions["SIMD_AVX512"] = is_x8
        cmake.definitions["SIMD_AVX512VNNI"] = is_x86
        cmake.definitions["SIMD_SHARED"] = self.options.shared
        cmake.definitions["SIMD_TEST"] = False
        return cmake

    def build(self):
        if self.settings.os == "Windows":
            ...
        else:
            cmake = self._configure_cmake()
            cmake.configure(source_folder="Simd/prj/cmake")
            cmake.build()
```

armv7-common-toolchain.cmake:

```
set(CMAKE_SYSTEM_NAME Linux)
set(CMAKE_SYSTEM_PROCESSOR armv7)
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
```

conan_home.jinja:

```
conan_home = {{ os.getenv("CONAN_USER_HOME", os.path.expanduser("~"))}}
cmake_toolchain_files = $conan_home/.conan/cmake_toolchain_files
```

target-profile:

```
include(conan_home.jinja)
cmake_toolchain_file=$cmake_toolchain_files/armv7-common-toolchain.cmake
[conf]
tools.build:sysroot=$sysroot
tools.cmake.cmaketoolchain:user_toolchain=["$cmake_toolchain_file"]
```

Conan Host профиль

```
toolchain_path=$iv_toolchain_root_dir/poky/oecore-x86_64/sysroots/x86_64-oesdk-
linux/usr/bin/arm-oemllib32-linux-gnueabi
toolchain_prefix=arm-oemllib32-linux-gnueabi
sysroot="$iv_toolchain_root_dir/poky/oecore-x86_64/sysroots/aarch64-oe-linux"
cc_cxx_flags_str="-mfpu=neon -mfloat-abi=softfp"

[env]
# [env] is deprecated! Use [buildenv] instead
CC="$toolchain_path/$toolchain_prefix-gcc --sysroot=$sysroot"
CXX="$toolchain_path/$toolchain_prefix-g++ --sysroot=$sysroot"
AS="$toolchain_path/$toolchain_prefix-as"
CXXFLAGS="$cc_cxx_flags_str"
CFLAGS="$cc_cxx_flags_str -std=gnu99"

xz_utils:CPP="$toolchain_path/$toolchain_prefix-cpp --sysroot=$sysroot"
xz_utils:CPPFLAGS="$cc_cxx_flags_str"
ffmpeg:AS="$toolchain_path/$toolchain_prefix-gcc --sysroot=$sysroot"
libpng:CFLAGS="$cc_cxx_flags_str --sysroot=$sysroot"
```

https://docs.conan.io/1/reference/profiles.html#package-settings-and-env-vars

Conan Host профиль

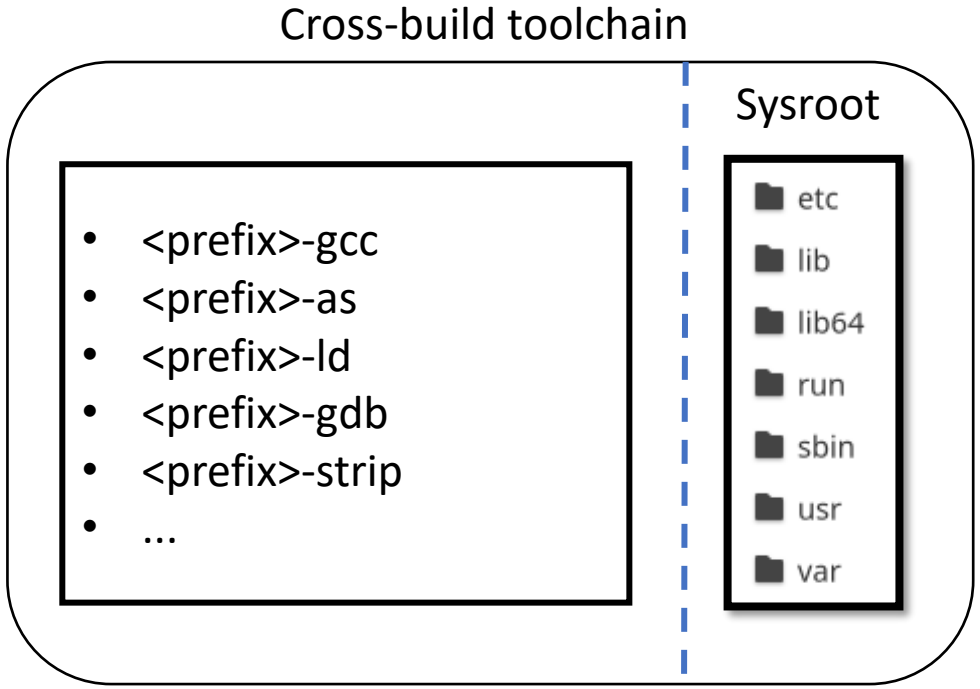
- Вспомогательные переменные:
  - x1=1
    x2=2_$x1
    x3=3_$x2
  - Ожидание: $x3 == 3_2_1
  - Реальность: $x3 == 3_2_$x1

```
toolchain_path=$iv_toolchain_root_dir/poky/oecore-x86_64/sysroots/x86_64-oesdk-linux/usr/bin/arm-oemllib32-linux-gnueabi
toolchain_prefix=arm-oemllib32-linux-gnueabi
sysroot="$iv_toolchain_root_dir/poky/oecore-x86_64/sysroots/aarch64-oe-linux"
cc_cxx_flags_str="-mfpu=neon -mfloat-abi=softfp"

[env]
CC="$toolchain_path/$toolchain_prefix-gcc --sysroot=$sysroot"
CXX="$toolchain_path/$toolchain_prefix-g++ --sysroot=$sysroot"
```

https://docs.conan.io/1/reference/profiles.html#variable-declaration

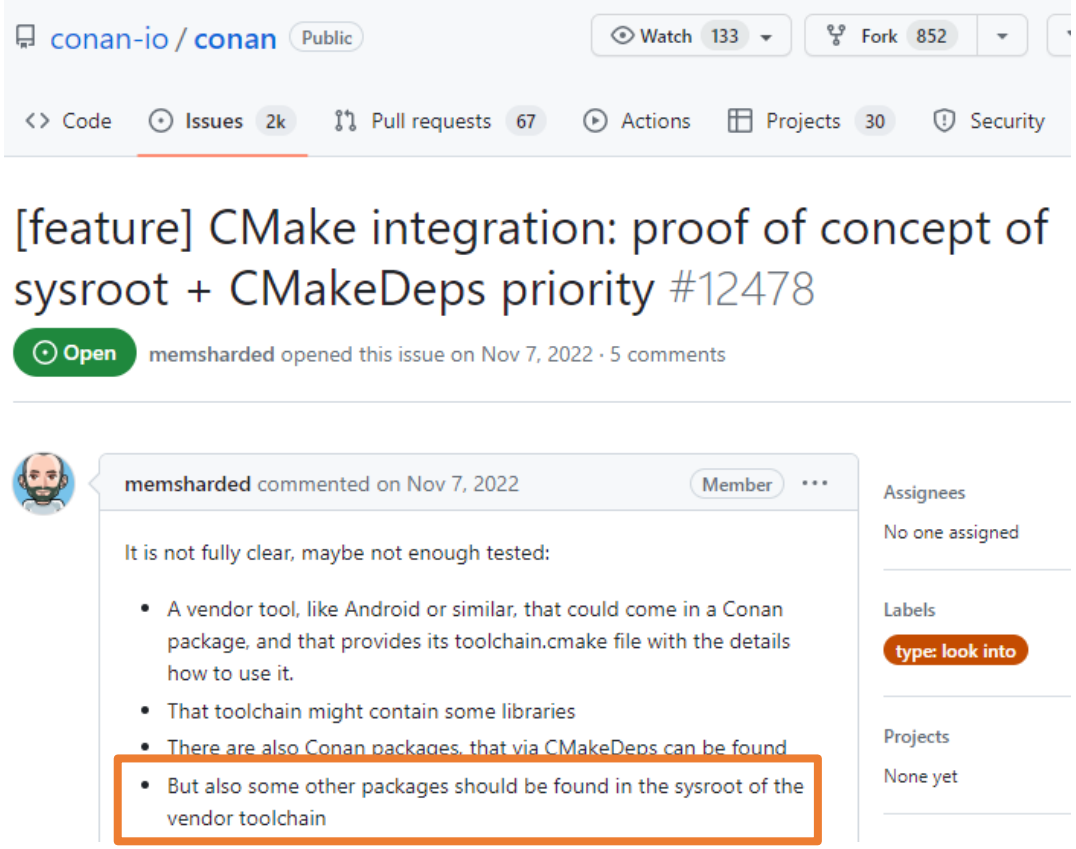
Conan и Sysroot

- Тулчейн:
  - Набор утилит для сборки и отладки для целевой платформы.
  - Заголовочные файлы для C/C++ библиотек.
  - Библиотеки (libc, стандартная библиотека C++ и др.) для целевой платформы.
- Sysroot:
  - Позволяет указать префикс для поиска заголовочных файлов и библиотек для целевой платформы.



https://gcc.gnu.org/onlinedocs/gcc/Directory-Options.html#index-sysroot

Проблемы Conan Sysroot



conan-io / conan Public

Watch 133 · Fork 852

Code · Issues 2k · Pull requests 67 · Actions · Projects 30 · Security

[feature] CMake integration: proof of concept of sysroot + CMakeDeps priority #12478

Open memsharded opened this issue on Nov 7, 2022 · 5 comments

memsharded commented on Nov 7, 2022 Member

It is not fully clear, maybe not enough tested:

- A vendor tool, like Android or similar, that could come in a Conan package, and that provides its toolchain.cmake file with the details how to use it.
- That toolchain might contain some libraries
- There are also Conan packages, that via CMakeDeps can be found
- But also some other packages should be found in the sysroot of the vendor toolchain

Assignees
No one assigned

Labels
type: look into

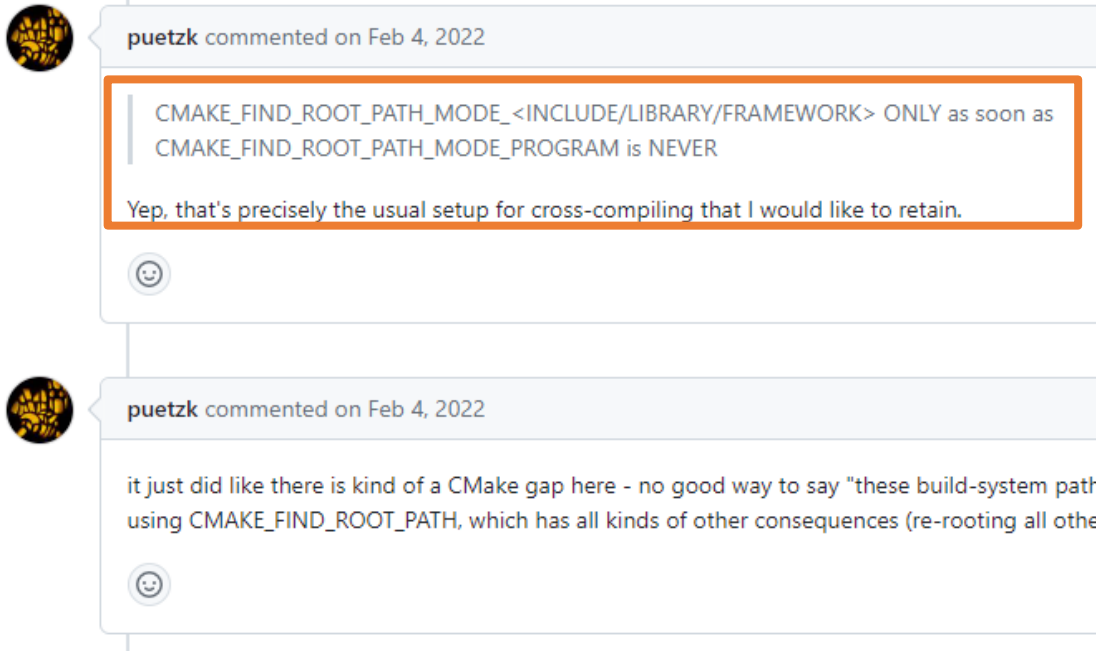
Projects
None yet

First of all, the solution we have found for us: For every oss library, which is part of our dependency graph and is available as a conan package in conan center as well as in the vendor sysroot, we have created a new conanfile. This conanfile only packages the prebuild library from that sysroot and we set the version to something generic. If we use zlib as an example, we name it for example: zlib/sysroot. So there are basically two conanfiles, which exists in parallel satisfying the same dependency. This zlib/sysroot package is now set as a requirement in the sysroot conan package. In this way we achieve to force any consumer of this library to override the zlib dependency, when there is another package in the graph using zlib. Our product application conan packages looks then basically like this:

```
def requirements (self):
    self.requires("opencv/4.5.5")
    ....
    if self.settings.arch == "armv8":
        self.requires("vendor/sysroot")
        self.requires("zlib/sysroot", override = True)
```

https://github.com/conan-io/conan/issues/12478
https://github.com/conan-io/conan/issues/12597

Проблемы Conan Sysroot



puetzk commented on Feb 4, 2022

> CMAKE_FIND_ROOT_PATH_MODE_<INCLUDE/LIBRARY/FRAMEWORK> ONLY as soon as CMAKE_FIND_ROOT_PATH_MODE_PROGRAM is NEVER

Yep, that's precisely the usual setup for cross-compiling that I would like to retain.

puetzk commented on Feb 4, 2022

it just did like there is kind of a CMake gap here - no good way to say "these build-system path using CMAKE_FIND_ROOT_PATH, which has all kinds of other consequences (re-rooting all othe

qwertzui11 commented on Jan 2, 2022 — Author

updated version, conan 1.45

Thx for your great answer.

However to my knowledge the variables (`CONAN_CMAKE_FIND_ROOT_PATH` and `CONAN_CMAKE_SYSROOT`) only apply to the "old" cmake build_helper tool.

I could not find any of them in the new `CMakeToolChain` implementation

There seems to be no SIMPLE way to set `CMAKE_FIND_ROOT_PATH` or `CMAKE_SYSROOT`. Am I wrong?

My current solution involves a "custom toolchain" with the following content:

```
toolchain_content = f"""
  set(CMAKE_FIND_ROOT_PATH {sysroot_folder})
  set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
  set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
  set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
  """
```

Due to the common case of setting a sysroot when cross-compiling I wonder, if I'm doing something wrong, because this should be easier?

Thx for your time!

https://github.com/conan-io/conan/issues/10513
https://github.com/conan-io/conan/issues/9368
https://github.com/conan-io/conan/issues/11380

opencv/4.5.5

- Рецепт из conan-center-index до поддержки Conan 2.0 (conans.CMake).
- Тулчейн от производителя на основе Yocto Project:
  - aarch64-qualcomm-gcc8.2-linux (poky2.6.4)

```
from conan.tools.microsoft import msvc_runtime_flag
from conans import ConanFile, CMake, tools
from conans.errors import ConanInvalidConfiguration
import os
import textwrap

required_conan_version = ">=1.43.0"


class OpenCVConan(ConanFile):
    name = "opencv"
    license = "Apache-2.0"
    homepage = "https://opencv.org"
    description = "OpenCV (Open Source Computer Vision Library)"
    url = "https://github.com/conan-io/conan-center-index"
    topics = ("computer-vision", "deep-learning", "image-processing")

    settings = "os", "arch", "compiler", "build_type"
```

opencv/4.5.5

```
include(packages/packages_common)
[settings]
os                = Linux
arch              = armv8
compiler          = gcc
compiler.version  = 8
compiler.libcxx   = libstdc++11
build_type        = Release

[options]
opencv:with_gtk = False
```

```
[options]
opencv:contrib=True
opencv:shared=False
opencv:with_jpeg=libjpeg-turbo
opencv:with_ffmpeg=True
opencv:with_ipp=False
```

opencv/4.5.5

```
[env]
...
CC="$toolchain_path/$toolchain_prefix-gcc"
CXX="$toolchain_path/$toolchain_prefix-g++"
```

```
  The C compiler


    "/opt/toolchains/poky/2.6.4/sysroots/x86_64-oesdk-linux/usr/bin/aarch64-oe-
linux/aarch64-oe-linux-gcc"


  is not able to compile a simple test program.
```

opencv/4.5.5

```
[env]
...
CC="$toolchain_path/$toolchain_prefix-gcc"
CXX="$toolchain_path/$toolchain_prefix-g++"
CONAN_CMAKE_SYSROOT=$sysroot
```

```
[ 24%] Building CXX object source_subfolder/modules/reg/CMakeFiles/opencv_reg.dir/src/mapper.cpp.o
[ 24%] Running cpp protocol buffer compiler on /home/user/.conan/data/opencv/4.5.5/conan/stable/build/762ac2
cea2bdd999098cd3ef38e96c2bf7f48992/source_subfolder/modules/dnn/src/tensorflow/versions.proto. Custom option
s:
/bin/sh: line 1: /opt/toolchains/poky/2.6.4/sysroots/aarch64-oe-linux/usr/bin/protoc: cannot execute binary
file: Exec format error
make[2]: *** [source_subfolder/modules/dnn/CMakeFiles/opencv_dnn.dir/build.make:147: source_subfolder/module
s/dnn/versions.pb.h] Error 126
make[1]: *** [CMakeFiles/Makefile2:3367: source_subfolder/modules/dnn/CMakeFiles/opencv_dnn.dir/all] Error 2
make[1]: *** Waiting for unfinished jobs....
```

opencv/4.5.5

```
[env]
...
CC="$toolchain_path/$toolchain_prefix-gcc"
CXX="$toolchain_path/$toolchain_prefix-g++"
CONAN_CMAKE_SYSROOT=$sysroot
CONAN_CMAKE_FIND_ROOT_PATH_MODE_PROGRAM=NEVER
```

-- Found WebP: /opt/toolchains/poky/2.6.4/sysroots/aarch64-oe-linux/usr/lib/libwebp.so

opencv/4.5.5

```
[env]
...
CC="$toolchain_path/$toolchain_prefix-gcc --sysroot=$sysroot"
CXX="$toolchain_path/$toolchain_prefix-g++ --sysroot=$sysroot"
# Если нужен CONAN_CMAKE_SYSROOT, то можно так
# opencv:CONAN_CMAKE_SYSROOT=$sysroot
```

-- Found WebP: /home/user/.conan/data/libwebp/1.2.2/conan/stable/package/<hash>/lib/libwebp.a

libtiff/4.3.0

- Рецепт из conan-center-index с поддержкой Conan 2.0 (conan.tools.cmake)
- Тулчейн производителя на основе Yocto Project:
  - aarch64-qualcomm-gcc8.2-linux (poky2.6.4)

```
from conan import ConanFile
from conan.errors import ConanInvalidConfiguration
from conan.tools.cmake import CMake, CMakeDeps, CMakeToolchain, cmake_layout
from conan.tools.files import apply_conandata_patches, copy, export_conandata
from conan.tools.microsoft import is_msvc
from conan.tools.scm import Version
import os

required_conan_version = ">=1.53.0"


class LibtiffConan(ConanFile):
    name = "libtiff"
    description = "Library for Tag Image File Format (TIFF)"
    url = "https://github.com/conan-io/conan-center-index"
    license = "MIT"
    homepage = "http://www.simplesystems.org/libtiff"
    topics = ("tiff", "image", "bigtiff", "tagged-image-file-format")

    settings = "os", "arch", "compiler", "build_type"
    options = {
        "shared": [True, False],
        "fPIC": [True, False],
        "lzma": [True, False],
        "jpeg": [False, "libjpeg", "libjpeg-turbo", "mozjpeg"],
        "zlib": [True, False],
        "libdeflate": [True, False],
        "zstd": [True, False],
        "jbig": [True, False],
        "webp": [True, False],
        "cxx":  [True, False],
```

libtiff/4.3.0

```
[env]
...
CC="$toolchain_path/$toolchain_prefix-gcc --sysroot=$sysroot"
CXX="$toolchain_path/$toolchain_prefix-g++ --sysroot=$sysroot"
```

```
<…>/gcc/aarch64-oe-linux/8.2.0/real-ld: skipping incompatible /usr/lib/libm.so.6 when searching for /usr/lib/libm.so.6
<…>/gcc/aarch64-oe-linux/8.2.0/real-ld: cannot find /usr/lib/libm.so.6
```

libtiff/4.3.0

```
[conf]
tools.build:sysroot=$sysroot
#или так "libtiff/*:tools.build:sysroot=$sysroot"

[env]
CC="$toolchain_path/$toolchain_prefix-gcc --sysroot=$sysroot"
CXX="$toolchain_path/$toolchain_prefix-g++ --sysroot=$sysroot"
```

libpng/1.6.38

- Рецепт из conan-center-index с поддержкой Conan 2.0 (conan.tools.cmake)
- Тулчейн производителя на основе Yocto Project:
  - aarch64-qualcomm-gcc8.2-linux (poky2.6.4)

```
from conan import ConanFile
from conan.errors import ConanInvalidConfiguration
from conan.tools.apple import is_apple_os
from conan.tools.build import cross_building
from conan.tools.cmake import CMake, CMakeToolchain, CMakeDeps, cmake_layout
from conan.tools.files import apply_conandata_patches, copy, export_conandata
from conan.tools.microsoft import is_msvc
from conan.tools.scm import Version
import os

required_conan_version = ">=1.53.0"


class LibpngConan(ConanFile):
    name = "libpng"
    description = "libpng is the official PNG file format reference library."
    url = "https://github.com/conan-io/conan-center-index"
    homepage = "http://www.libpng.org"
    license = "libpng-2.0"
    topics = ("png", "graphics", "image")
    settings = "os", "arch", "compiler", "build_type"
    options = {
        "shared": [True, False],
        "fPIC": [True, False],
        "neon": [True, "check", False],
        "msa": [True, False],
        "sse": [True, False],
        "vsx": [True, False],
        "api_prefix": ["ANY"],
    }
```

libpng/1.6.38

```
[conf]
tools.build:sysroot=$sysroot

[env]
CC="$toolchain_path/$toolchain_prefix-gcc --sysroot=$sysroot"
CXX="$toolchain_path/$toolchain_prefix-g++ --sysroot=$sysroot"
```

```
/home/user/.conan/data/zlib/1.2.13/_/_/package/<hash>/include/zconf.h:452:14: fatal
error: sys/types.h: No such file or directory
#    include <sys/types.h>      /* for off_t */
             ^~~~~~~~~~~~~
CMake Error at scripts/genout.cmake:78 (message):
  Failed to generate
  /home/user/.conan/data/libpng/1.6.38/_/_/build/<hash>/build/Release/pnglibconf.out.tf1
```

libpng/1.6.38

```
[conf]
tools.build:sysroot=$sysroot

[env]
CC="$toolchain_path/$toolchain_prefix-gcc --sysroot=$sysroot"
CXX="$toolchain_path/$toolchain_prefix-g++ --sysroot=$sysroot"
```

```
execute_process(COMMAND "${CMAKE_C_COMPILER}" "-E"
                        ${CMAKE_C_FLAGS}
                        ${PLATFORM_C_FLAGS}
                        "-I${SRCDIR}"
                        "-I${BINDIR}"
                        ${INCLUDES}
                        "-DPNGLIB_LIBNAME=PNG${PNGLIB_MAJOR}${PNGLIB_MINOR}_0"
                        "-DPNGLIB_VERSION=${PNGLIB_VERSION}"
                        "-DSYMBOL_PREFIX=${SYMBOL_PREFIX}"
                        "-DPNG_NO_USE_READ_MACROS"
                        "-DPNG_BUILDING_SYMBOL_TABLE"
                        ${PNG_PREFIX_DEF}
                        "${INPUT}"
                OUTPUT_FILE "${OUTPUT}.tf1"
                WORKING_DIRECTORY "${BINDIR}"
                RESULT_VARIABLE CPP_FAIL)
if(CPP_FAIL)
  message(FATAL_ERROR "Failed to generate ${OUTPUT}.tf1")
endif()
```

libpng/1.6.38

```
[conf]
tools.build:sysroot=$sysroot

[env]
CC="$toolchain_path/$toolchain_prefix-gcc --sysroot=$sysroot"
CXX="$toolchain_path/$toolchain_prefix-g++ --sysroot=$sysroot“
libpng:CFLAGS="--sysroot=$sysroot"
```

Тулчейн в пакете Conan

```
def package_info(self):
    toolchain_path = self.package_folder + "/toolchain"
    self.env_info.PATH = [
        f"{toolchain_path}/arm-openwrt-linux-gnueabi/bin"]
    self.env_info.CC = f"{self.compiler_prefix}gcc --sysroot={self.sysroot}"
    self.env_info.CXX = f"{self.compiler_prefix}g++ --sysroot={self.sysroot}"
    self.env_info.CXXFLAGS = self.c_cxx_flags
    self.env_info.CFLAGS = self.c_flags
    self.env_info.STAGING_DIR = self.package_folder + "/staging_dir"
```

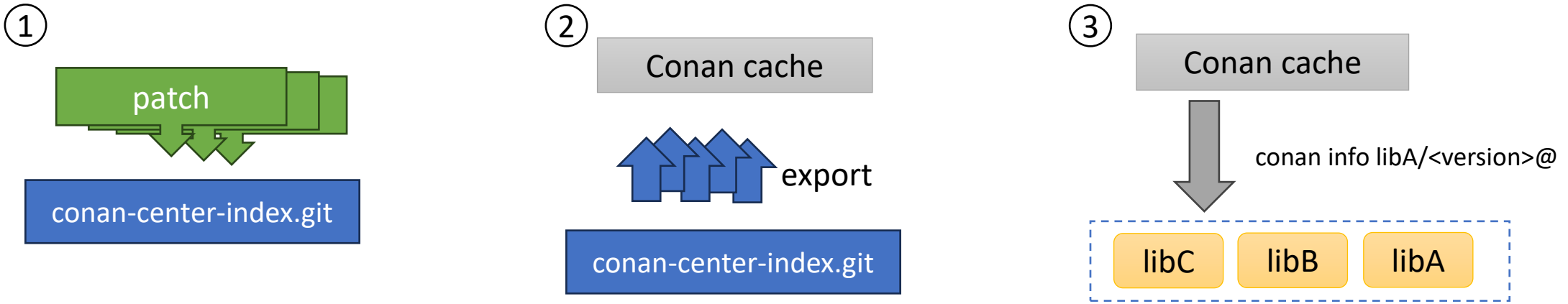
```
[tool_requires]
armv7-linux-openwrt-toolchain/0.1@iv/rc
```

Тулчейн в пакете Conan

```python
def package_info(self):
    self.conf_info.define("tools.cmake.cmaketoolchain:system_processor", "armv7")
    self.conf_info.define("tools.cmake.cmaketoolchain:system_name", "Linux")
    self.conf_info.define("tools.build:sysroot", self.sysroot)
    toolchain_path = self.package_folder + "/toolchain"
    self.buildenv_info.prepend_path("PATH", f"{toolchain_path}/arm-openwrt-linux-gnueabi/bin")
    self.buildenv_info.define("CC", f"{self.compiler_prefix}gcc --sysroot={self.sysroot}")
    self.buildenv_info.define("CXX", f"{self.compiler_prefix}g++ --sysroot={self.sysroot}")
    self.buildenv_info.define("CXXFLAGS", self.c_cxx_flags)
    self.buildenv_info.define("CFLAGS", self.c_flags)
    ...
```

- Conan >1.46:
  - self.conf_info (CMakeToolchain, AutotoolsToolchain)
  - self.env_info DEPRECATED
  - buildenv_info/runenv_info

"Зеркалирование" рецептов

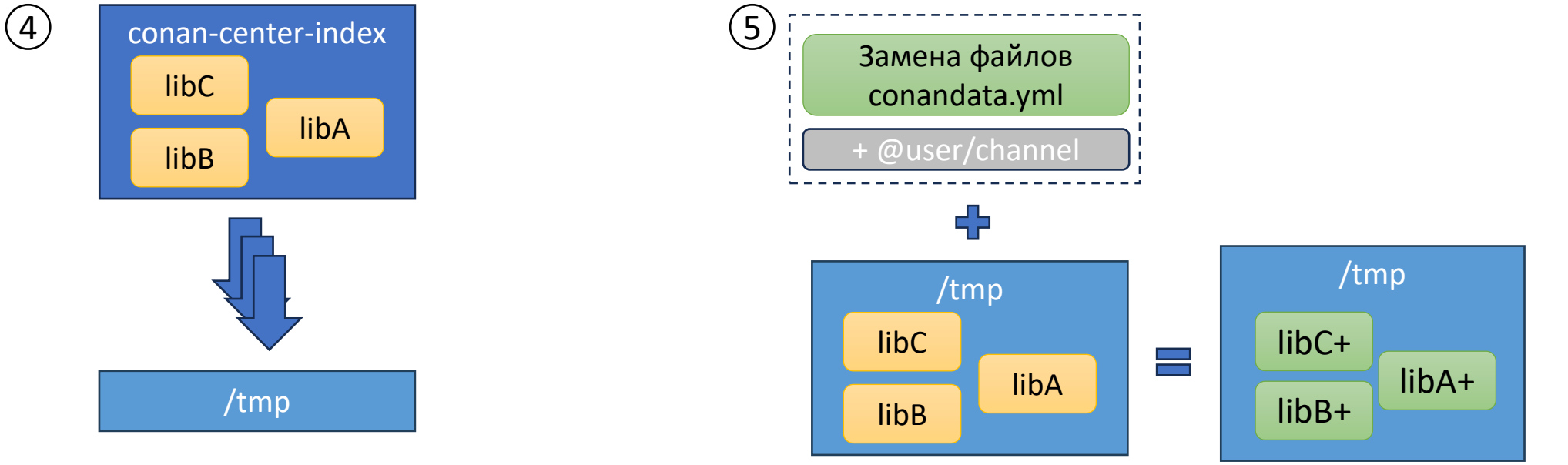


Исправляем рецепты если нужно

Предварительно экпортируем рецепты в локальный кеш Conan

Получаем список всех зависимостей для нужного референса или conanfile.txt

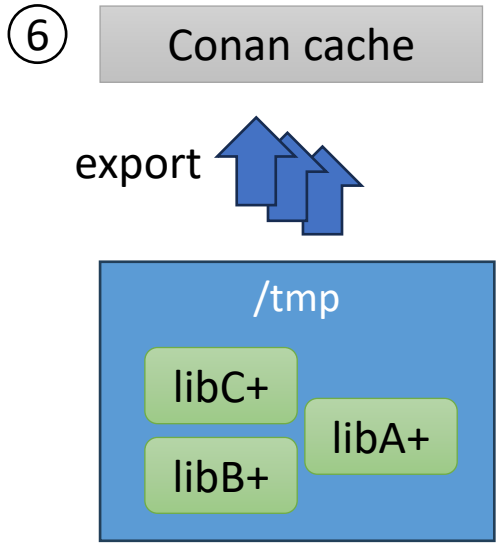
"Зеркалирование" рецептов

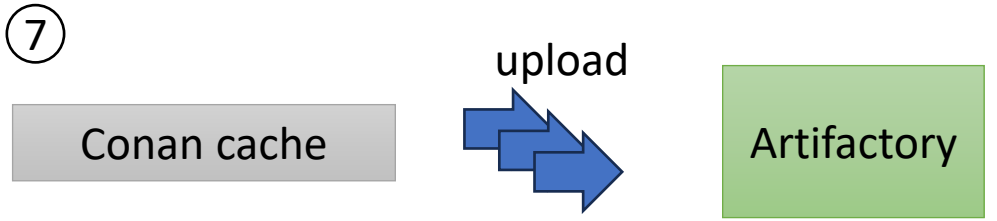


Копируем нужные рецепты во временную папку

Модифицируем conandata.yml на новые пути локального хранилища и заменяем @user/channel

"Зеркалирование" рецептов



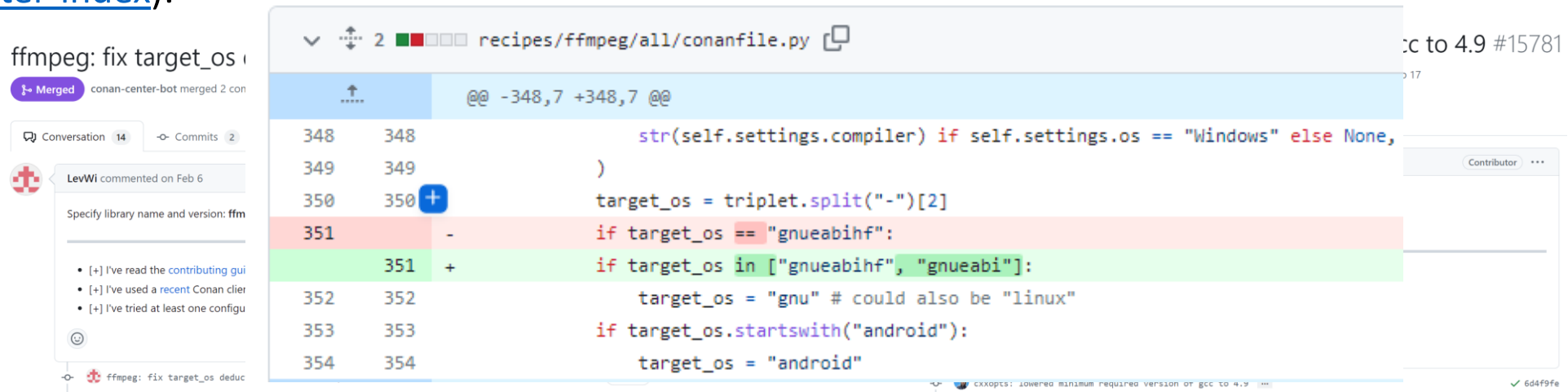


Экспортируем модифицированные рецепты в локальный кеш Conan

Загружаем модифицированные рецепты в центральный артефакторий

Участие в open source сообществе

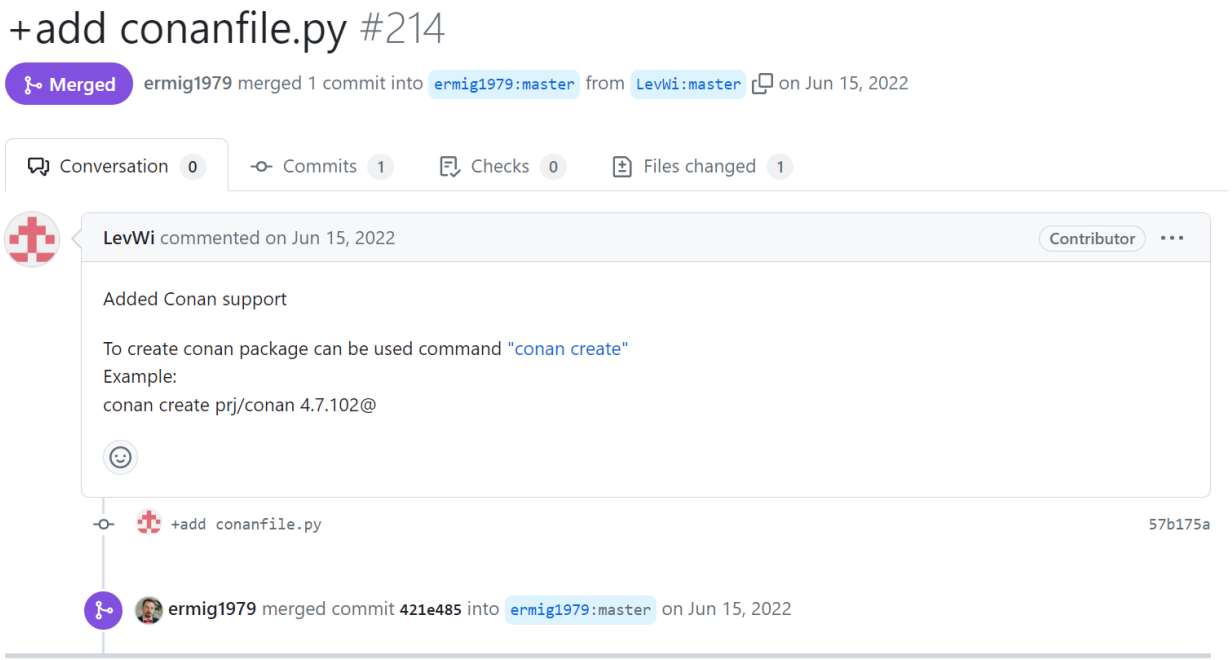
- Conan (https://github.com/conan-io/conan-center-index):
  - https://github.com/conan-io/conan-center-index/pull/15777



- https://github.com/conan-io/conan-center-index/pull/15781
- https://github.com/conan-io/conan-center-index/issues/10402
- https://github.com/conan-io/conan-center-index/issues/11924

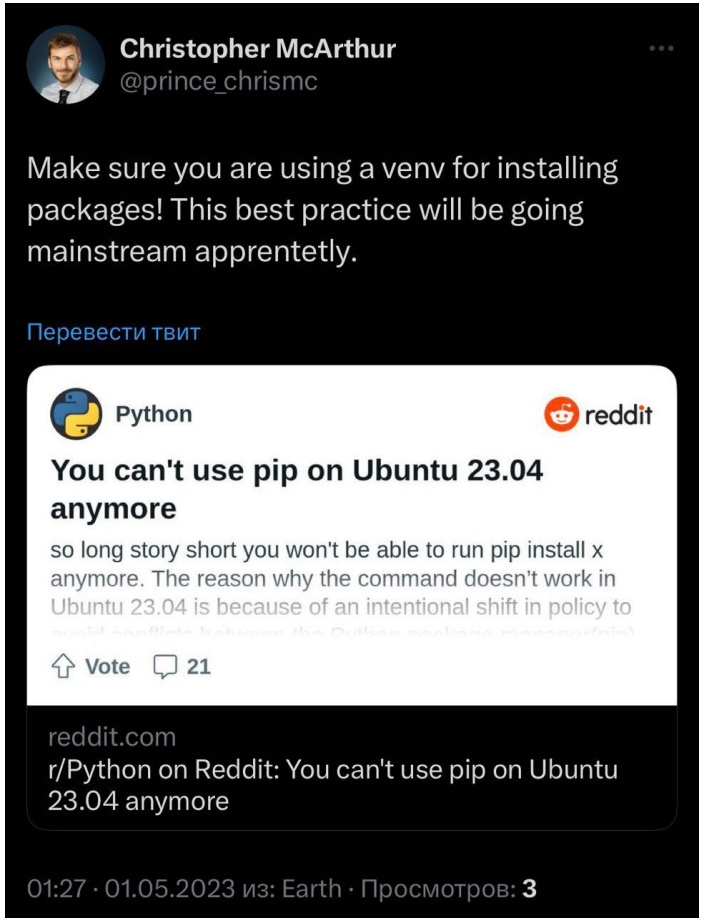
Участие в open source сообществе

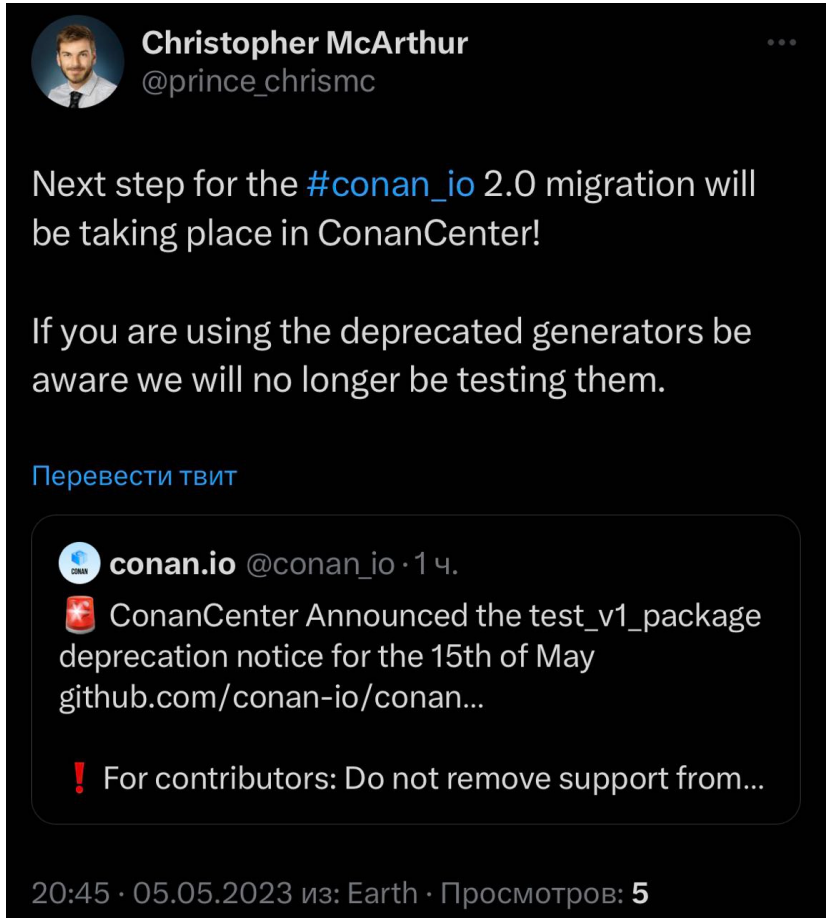
- Simd (https://github.com/ermig1979/Simd):



  - https://github.com/ermig1979/Simd/pull/214

Conan 2.0





Christopher McArthur - Conan Developer Advocate - JFrog

Что хотели получить

- Быстро обновляться на свежие версии пакетов.
- Легко добавлять новые зависимости.
- Легко портировать на новые платформы.
- Иметь воспроизводимые процессы построения.
- Экономить место на жестком диске.
- Иметь старые версии пакетов для воспроизводимости.
- Переиспользовать кеши на билд машинах.
- При необходимости собирать из исходных кодов.
- Собранные пакеты должны быть протестированы.
- Скачивать пакеты из локального артефактория.

Что удалось получить

- Быстрый и простой процесс настройки для разработчика.
- Прозрачность, просто работаете с CMake проектом.
- Экономия места.
- Вопросы к поддержке:
  - Сложно сменить версию зависимости.
  - Не просто перестроить под все платформы.
  - Трудно подружить с билд системой.

	Было	Стало
Артефакторий, Гб	482.3 (355)	33.8
FFmpeg, Гб	4.4	0.9
Время сборки, сек	90	45

Время для вопросов!

Семен Буденков

semen.budenkov@intelli-vision.com