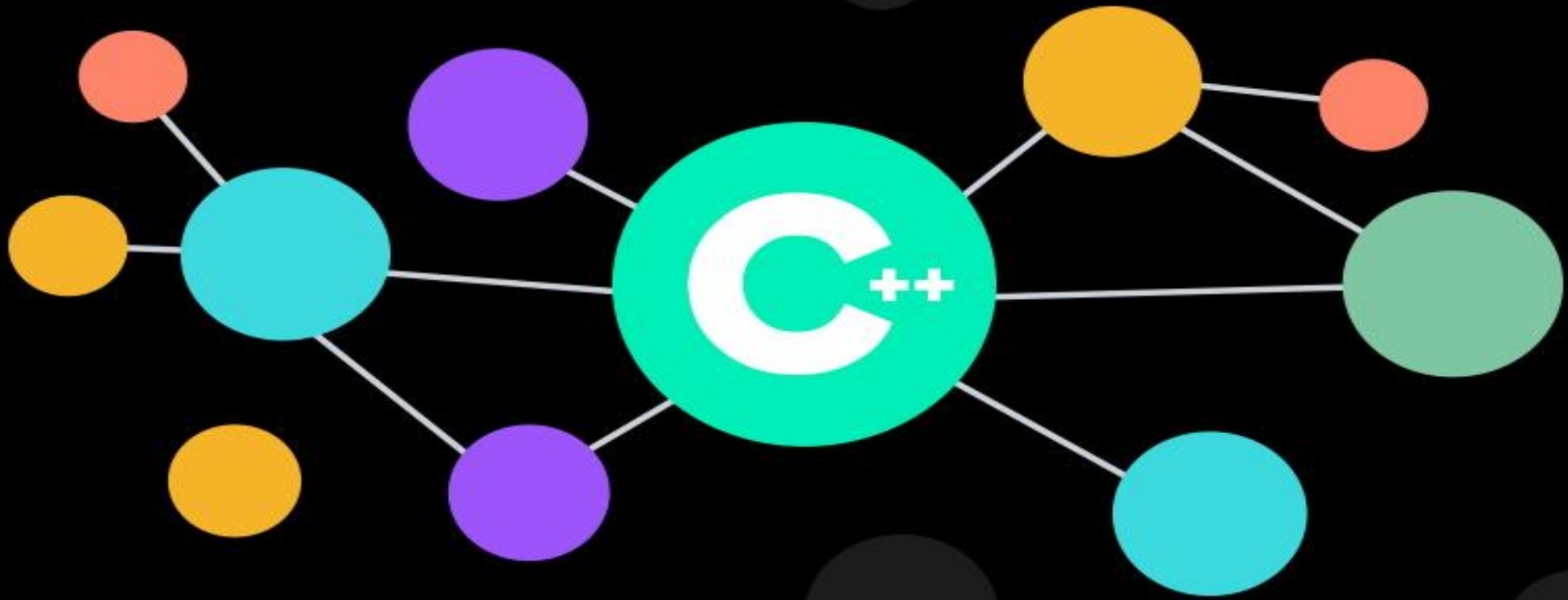

C++

Package management in C++

Mikhail Svetkin

NDC TechTown, 2022

About me

Senior software engineer at reMarkable

C++ programmer for last 11 years

Areas: embedded, networking, frameworks, libraries, build systems

Agenda

- What is package management
- Introduction to vcpkg
- Introduction to conan 1.x
- Summary

What is package manager in c++?

- Tool (or collection of tools)
- Simplify: search, install, configure, upgrade, remove **libraries (dependencies)**
- Different kind of **artifacts**

C++ package manager or dependency manager?

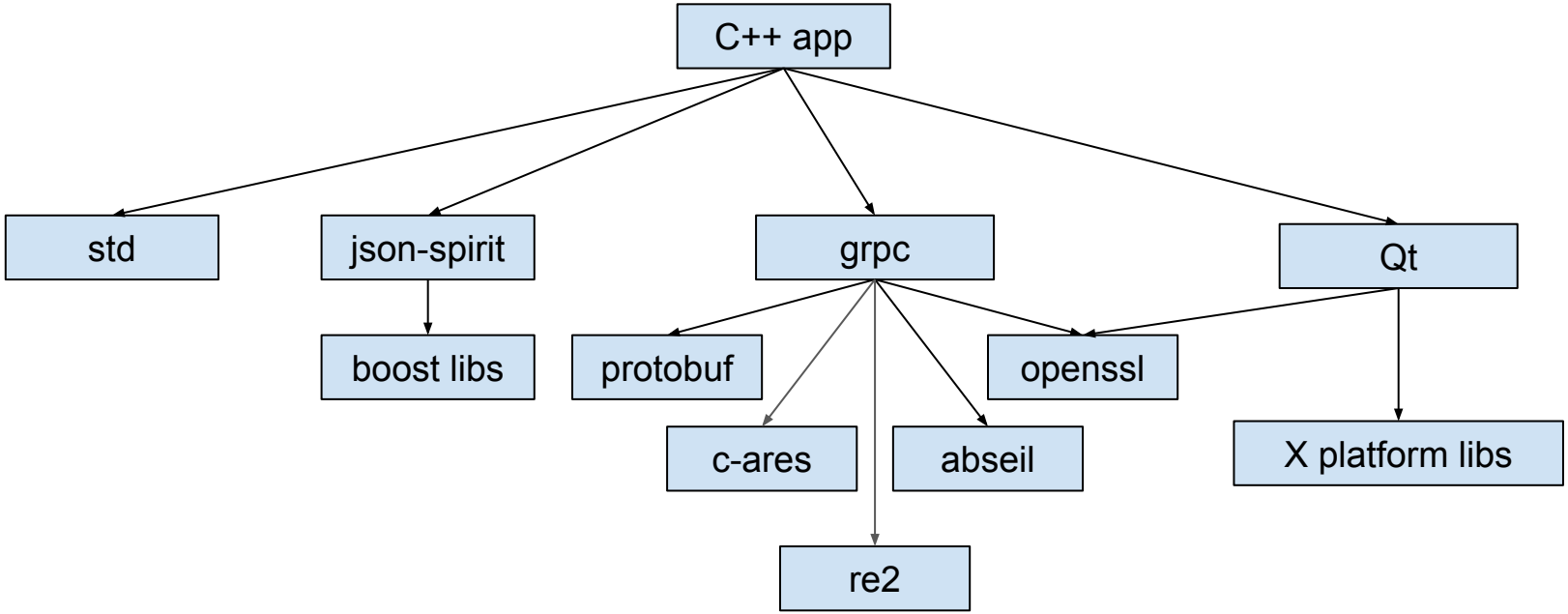
About vcpkg

vcpkg is a free C/C++ package manager

Conan, the C/C++ Package Manager

Conan is a **dependency** and package **manager** for C and C++ languages.

Dependencies in real life



Dependency counter: 1[redacted] + X

Wiki page

For other distros, get the separate components below.

Build essentials

Ubuntu and/or Debian:	`sudo apt-get install build-essential perl python3 git`
Fedora 30	`su - -c "dnf install perl-version git gcc-c++ compat-openssl10-devel harfbuzz-devel double-conversion-devel libzstd-devel at-spi2-atk-devel dbus-devel mesa-libGL-devel"`
OpenSUSE:	`sudo zypper install git-core gcc-c++ make`

Libxcb

Libxcb is now the default window-system backend for platforms based on X11/Xorg, and you should therefore install libxcb and its accompanying packages. Qt5 should build with whatever libxcb version is available in your distro's packages (but you may optionally wish to use v1.8 or higher to have threaded rendering support). The README lists the required packages.

Ubuntu/Debian:	`sudo apt-get install '^libxcb.*-dev' libx11-xcb-dev libglu1-mesa-dev libxrender-dev libxi-dev libxkbcommon-dev libxkbcommon-x11-dev`
Fedora 30:	`su - -c "dnf install libxcb libxcb-devel xcb-util xcb-util-devel xcb-util-*-devel libX11-devel libXrender-devel libxkbcommon-devel libxkbcommon-x11-devel libXi-devel libdrm-devel libXcursor-devel libXcomposite-devel"`
OpenSUSE 12+:	`sudo zypper in xorg-x11-libxcb-devel xcb-util-devel xcb-util-image-devel xcb-util-keysyms-devel xcb-util-renderutil-devel xcb-util-wm-devel xorg-x11-devel libxkbcommon-x11-devel libxkbcommon-devel libXi-devel`
ArchLinux/Manjaro:	`sudo pacman -S --needed libxcb xcb-proto xcb-util xcb-util-image xcb-util-wm libxi`
Chakra Linux:	Install the ArchLinux packages, plus xcb-util-keysyms. It's available from CCR.
Mandriva/ROSA/Unity:	`urpmi 'pkgconfig(xcb)' 'pkgconfig(xcb-icccm)' 'pkgconfig(xcb-image)' 'pkgconfig(xcb-renderutil)' 'pkgconfig(xcb-keysyms)' 'pkgconfig(xrender)'`
Linux Mint:	`apt-get install libx11-xcb-dev libxcb-composite0-dev libxcb-cursor-dev libxcb-damage0-dev libxcb-dpms0-dev libxcb-dri2-0-dev libxcb-dri3-dev libxcb-glx0-dev libxcb-icccm4-dev libxcb-image0-dev libxcb-keysyms1-dev libxcb-present-dev libxcb-randr0-dev libxcb-render-util0-dev libxcb-render0-dev libxcb-shape0-dev libxcb-shm0-dev libxcb-sync-dev libxcb-util-dev libxcb-xfixes0-dev libxcb-xinerama0-dev libxcb-xkb-dev libxcb-xtest0-dev libxcb1-dev`

Centos 5/6	Install missing Qt build dependencies:
	`yum install libxcb libxcb-devel xcb-util xcb-util-devel`
	Install Red Hat DevTools 1.1 for CentOS-5/6 x86_64, they are required due to outdated GCC shipped with default CentOS:
	`wget http://people.centos.org/tru/devtools-1.1/devtools-1.1.repo -O /etc/yum.repos.d/devtools-1.1.repo`
	`yum install devtoolset-1.1`
	Initialise your newly installed dev tools:
	`scl enable devtoolset-1.1 bash`
	`# Test - Expect to see gcc version 4.7.2 (not gcc version 4.4.7)`
	`gcc -v`
	For more info on preparing the environment on **CentOS**, see this thread.

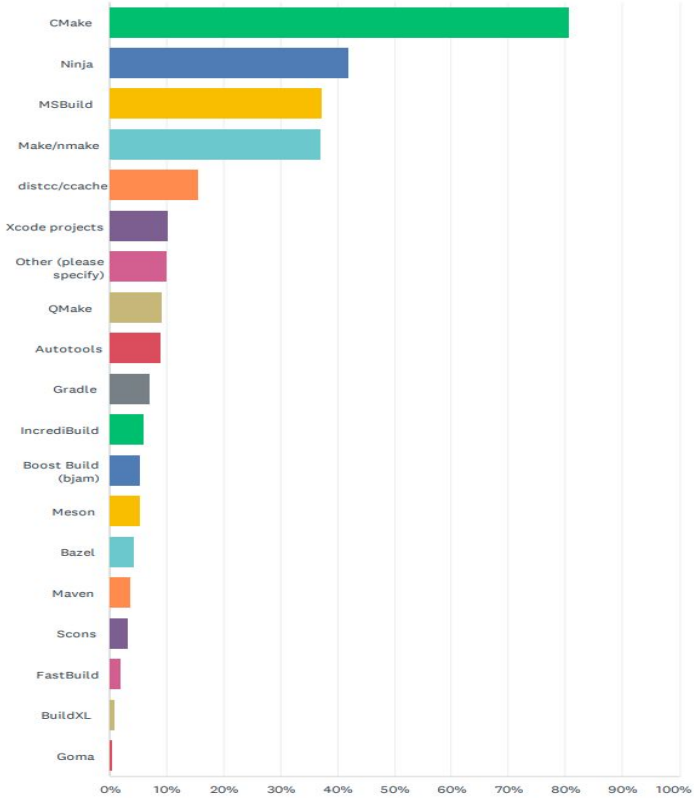
Centos 7	Update to gcc 7:
	`yum install centos-release-scl`
	`yum install devtoolset-7-gcc*`
	`scl enable devtoolset-7 bash`
	Install missing Qt build dependencies (Qt 5.13):
	`yum install libxcb libxcb-devel xcb-util xcb-util-devel mesa-libGL-devel libxkbcommon-devel`

https://wiki.qt.io/Building_Qt_5_from_Git

Dependencies as subdirectory

- Each project has copy of this dependencies
- Have to be (re)compiled per each project or clean build
- Hard to test against different versions of dependencies
- Multiple copies of the same dependencies take disk space
- CMake: targets with the same names may collide between projects

2022 Annual C++ Developer Survey "Lite"



Build your own solution

- You have/had your arguments to do that

Package managers

~now days:

- connan - 1354
- vcpkg - 1910
- cargo - 89,786
- python - 140,000
- node   - 600,000

2018:

- conan - 340 (non confirmed)
- vcpkg - 800
- cargo - 1800

Why is that hard?

- Build systems (autotools, make, CMake, SCons, Meson, MSBuild, Waf, …)
- Compilers (gcc-X, clang-X, MSVC-X, …)
- C++ standard library (libstdc++, libstdc++11, libc++)
- C++ standard version (c++98/03/11/14/17/20/23/…)
- Project preprocessor flags
- Platform specific flags

What do we want from c++ package manager?

- Simple integration
- Download/Configure/Build/reuse
- Cross-compilation
- Version managing
- Custom registry
- Configure developer environment
- Ideally no extra deps

Introduction to vcpkg

vcpkg

- Standalone
- Portable
- Open-source
- Written in C++
- By Microsoft
- Contains ports only

How do I get vcpkg?

- Clone https://github.com/microsoft/vcpkg.git and invoke bootstrap-vcpkg.(sh|bat)
- Use one for the scripts: ". <(curl https://aka.ms/vcpkg-init.sh -L)"

Command line mode

```
vcpkg search fmt
vcpkg install fmt
vcpkg install fmt:x64-linux-dynamic
vcpkg install fmt:arm-linux
vcpkg depend-info fmt
vcpkg remove fmt
```

Manifest mode - vcpkg.json

```
{
  "name": "vcpkg-manifest-simple",
  "version-string": "0.0.1",
  "dependencies": [
    "fmt",
    "grpc"
  ]
}
```

Manifest mode - vcpkg.json

```
{
  "name": "vcpkg-manifest-extended" ,
  "version-string": "0.0.1",
  "dependencies": [
    "fmt",
    { "name" : "grpc", "version>=" : "1.48.0" }
  ],
  "builtin-baseline": "1c60450e5a4b328ad811a63030234cb0a7a19bb4" ,
  "overrides": [
    { "name": "fmt", "version": "8.1.0" }
  ]
}
```

CMake integration

```
cmake -B [build directory] -S . \
      -DCMAKE_TOOLCHAIN_FILE=[path to
vcpkg]/scripts/buildsystems/vcpkg.cmake
cmake --build [build directory]
```

CMake integration with presets

```
{
  "version": 4,
  "cmakeMinimumRequired" : {
    "major": 3,
    "minor": 23,
    "patch": 0
  },
  "configurePresets" : [
    {
      "name": "default",
      "displayName" : "Default Config",
      "description" : "Default build using Ninja generator",
      "generator" : "Ninja",
      "toolchainFile": "$env{VCPKG_ROOT}/scripts/buildsystems/vcpkg.cmake"
    }
  ],
  "buildPresets" : [
    {
      "name": "default",
      "configurePreset" : "default"
    }
  ],
  "testPresets" : [
    {
      "name": "default",
      "configurePreset" : "default",
      "output": {"outputOnFailure" : true},
      "execution" : {"noTestsAction" : "error", "stopOnFailure" : true}
    }
  ]
}
```

CMake integration with presets

```
cmake --preset default
cmake --build --preset default
```

Build configurations

X64-linux-dynamic.cmake - triplet file

```
set(VCPKG_TARGET_ARCHITECTURE x64)
set(VCPKG_CRT_LINKAGE dynamic)
set(VCPKG_LIBRARY_LINKAGE dynamic)
set(VCPKG_CMAKE_SYSTEM_NAME Linux)
set(VCPKG_FIXUP_ELF_RPATH ON)
```

```
cmake -B [build directory] -S . \
   -DCMAKE_TOOLCHAIN_FILE=[path to vcpkg]/scripts/buildsystems/vcpkg.cmake
   -DVCPKG_TARGET_TRIPLET=x64-linux-dynamic
   -DVCPKG_OVERLAY_TRIPLET=[path to custom triplets]
```

Main options to cmake:

- -DVCPKG_TARGET_TRIPLET
- -DVCPKG_HOST_TRIPLET
- -DVCPKG_OVERLAY_TRIPLETS
- -DVCPKG_OVERLAY_PORTS

How to add a package?

- Fork https://github.com/microsoft/vcpkg
  - add a new port
  - make sure that all developers uses your fork
  - or send a pull request
- Create a custom registry
  - add a new port
  - add vcpkg-configuration.json to the project

fmt/portfile.cmake

```
vcpkg_from_github(
    OUT_SOURCE_PATH SOURCE_PATH
    REPO fmtlib/fmt
    REF 8.1.1
    SHA512 794a47d7cb352a2a9f2c050a60a46b002e4157e...
    HEAD_REF master
    PATCHES
        fix-write-batch.patch
        fix-invalid-command.patch
)

vcpkg_cmake_configure(
    SOURCE_PATH "${SOURCE_PATH}"
    OPTIONS
        -DFMT_CMAKE_DIR=share/fmt
        -DFMT_TEST=OFF
        -DFMT_DOC=OFF
)

vcpkg_cmake_install()

…

vcpkg_cmake_config_fixup()
vcpkg_fixup_pkgconfig()
```

vcpkg-configuration.json

```
{
  "registries": [
    {
      "kind": "git",
      "repository": "git@github.com..." ,
      "baseline": "1c60450e5a4b328ad811a63030234cb0a7a19bb4" ,
      "packages": [ "fmt" ]
    }
  ]
}
```

vcpkg-configuration.json with artifacts (experimental)

```
{
  "registries": [
    {
      "kind": "artifact",
      "name": "microsoft",
      "location": "https://aka.ms/vcpkg-ce-default "
    }
  ],
  "requires": {
    "microsoft:tools/ninja-build/ninja" : "* 1.10.2",
    "microsoft:tools/kitware/cmake" : "* 3.20.1"
  }
}
```

vcpkg cache results

```
$ cmake --preset vcpkg-simple
Total elapsed time: 7.753 min
```

```
$ cmake --preset vcpkg-simple
Total elapsed time: 3.494 sec
```

vcpkg hash calculation

- Every file in the port directory, triplet file and name
- The C/C++ compiler executable
- The set of features selected
- Every dependency package hash
- All helper scripts referenced by portfile.cmake
- The version of CMake used
- The hash of the toolchain file
- Windows: The contents of any environment variables listed in VCPKG_ENV_PASSTHROUGH

vcpkg caching

- Build dependencies (or fetch cached binaries)
- Optional: Wrap into NuGet packages or cloud object Storage
- Optional: Upload to a storage
- Re-use for a next build

vcpkg storage

- Any NuGet compatible instance
- Github packages
- Azure artifacts
- Azure blob
- Google cloud

Does vcpkg meets our needs?

- Integration: Simple
- Binary caching: local, remote (manual setup)
- Cross-compilation: yes
- Versions managing: yes
- Custom registry: yes
- Can download dev tools (some part experimental)
- Extra deps: no

Introduction to conan 1.x

Conan

- Portable
- Open-source
- Written in python
- By jfrog
- Contains recipes only

How do I get conan?

- `pip install conan (recommended way)`
- `Visit https://conan.io/downloads.html`

Configure and setup profile

```
conan profile new --detect default
cat ~/.conan/profile/default

[settings]
os=Linux
os build=Linux
arch=x86 64
arch build=x86_64
compiler=gcc
compiler.version=11
compiler.libcxx=libstdc++ --> libstdc++11
build_type=Release

[options]
SomeLib:shared=True

[build requires]
tools, utilities that only run at build-time


[env]
env_var=value
```

Command line mode

```
conan search fmt
conan install fmt (works only with conanfile.py or conanfile.txt)
conan info fmt
```

conanfile.txt

```
[requires]
fmt/9.0.0
grpc/1.47.0

[generators]
CMakeToolchain
CMakeDeps
```

conanfile.txt - extended

```
[requires]
fmt/8.0.1
grpc/[>=1.47.0]

[tool_requires]
7zip/16.00

[generators]
CMakeToolchain
CMakeDeps

[options]
fmt:shared=True
grpc:shared=True

[imports]
bin, *.dll -> ./bin
lib, *.dylib* -> ./bin
```

CMake integration

```
mkdir build && cd build
conan install ..
cmake .. -DCMAKE_TOOLCHAIN_FILE=conan_toolchain.cmake
cmake build .
```

conanfile.py

```
from conans import ConanFile, CMake

class AwesomeProjectConan(ConanFile):
  settings = "os", "compiler", "build_type", "arch"
  requires = ("fmt/9.0.0" , "grpc/1.48.0")
  generators = "CMakeToolchain", "CMakeDeps"

  def imports(self):
      self.copy("*.dll", dst="bin", src="bin")
      self.copy("*.dylib*", dst="bin", src="lib")

  def configure(self):
      self.options["fmt"].shared = True

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

  def layout(self):
  def package(self):
  def source(self):
  def build_requirements(self):
```

conanfile.py build

```
conan install . --install-folder build
conan build . --build-folder build
```

Other build systems integration

- msbuild
- qbs
- qmake
- make
- Xcode
- possibility to support any build system

conan cache benchmarking

```
$ conan install ..

ERROR: Missing prebuilt package for 'abseil/20211102.0', 'c-ares/1.18.1',
'fmt/9.0.0', 'googleapis/cci.20220711', 'grpc/1.48.0', 'grpc-proto/cci.20220627',
'openssl/1.1.1q', 'protobuf/3.21.4', 're2/20220601', 'zlib/1.2.12'
```

```
$ conan install .. --build=missing
Total elapsed time: 9.541 min
```

```
$ conan install ..
Total elapsed time: 0.827 sec
```

conan hash calculation

- Every settings value in used profile
- Every options value in used profile
- Every requirements value in used profile

conan caching

- Fetch pre-build / build dependencies
- Optional: Wrap into conan packages
- Optional: Upload to a storage
- Re-use for a next build

conan storage

- JFrog Artifactory (self managed, cloud, on-premise)

Does conan meets our needs?

- Integration: Simple enough with extra step
- Binary caching: local, remote (global artifactory)
- Cross-compilation: yes
- Versions managing: yes
- Custom registry: yes
- Can download dev tools
- Extra deps: python

Summary

vcpkg vs conan: generally

	vcpkg	conan
Simple to use?	Looks like yet another package manager	Generally yes, but It depends
Simple integration	with CMake	Generally yes, but it depends
Binary caching	Local, requires to setup remote	Local/Remote
Cross-compilation	Yes	Yes
Custom-registry	Yes	Yes
Can download dev tools	Yes, some parts experimental	Yes
Extra deps	no	python

vcpkg vs conan: CMake integration

```
cmake -B [build directory] -S . \
  -DCMAKE_TOOLCHAIN_FILE=[path to vcpkg.cmake]
cmake --build [build directory]
```

```
mkdir build && cd build
conan install ..
cmake .. \

-DCMAKE_TOOLCHAIN_FILE=conan_toolchain.cmake
cmake -build .
```

vcpkg vs conan: cache results

```
$ cmake --preset vcpkg-extended
Total elapsed time:  7.753 min
```

```
$ cmake --preset vcpkg-extended
Total elapsed time:  3.494 sec
```

```
$ mkdir build && cd build
$ conan install ..

ERROR: Missing prebuilt package for
'abseil/20211102.0', 'c-ares/1.18.1',
'fmt/9.0.0', 'googleapis/cci.20220711',
'grpc/1.48.0', 'grpc-proto/cci.20220627',
'openssl/1.1.1q', 'protobuf/3.21.4',
're2/20220601', 'zlib/1.2.12'
```

```
$ conan install .. --build=missing
Total elapsed time:  9.541 min
```

```
$ conan install ..
Total elapsed time:  0.827 sec
```

Conclusion

- It is better to have a package managers
- Vcpkg is simpler and less mature / flexible in some parts
- Conan provides better control and has API for extensions
- Choose which one works for you

Links

- CMake + Conan: 3 Years Later - Mateusz Pusz - [CppNow 2021]
- Dependency Management in C++ - Patricia Aas - NDC TechTown 2021
- Dependency management in C++ - Xavier Bonaventura - code::dive 2019
- CppCon 2018: “C++ Dependency Management: from Package Consumption to Project Development”
- Conan and Conan Center: 2021 in numbers
- 2021 Annual C++ Developer Survey "Lite"
- 2022 Annual C++ Developer Survey "Lite"
- https://www.jetbrains.com/lp/devecosystem-2021/cpp/

Thank you
and questions?

Test project: https://github.com/msvetkin/ndctechtown2022

Email: mikhail.svetkin@gmail.com  https://t.me/msvetkin