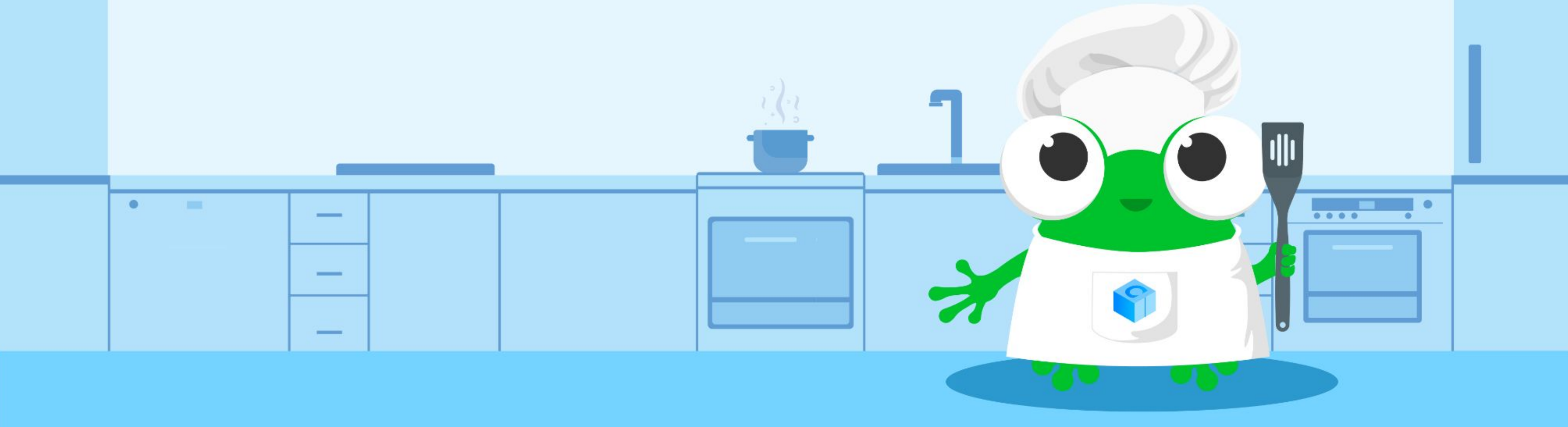
CONAN | JFrog

WHAT'S NEW IN CONAN 2.0

The lessons we have learned from the C++ ecosystem

Christopher McArthur, Conan Developer Advocate



JFrog

CONAN
C/C++ Package Manager

Everything is new!

1.0

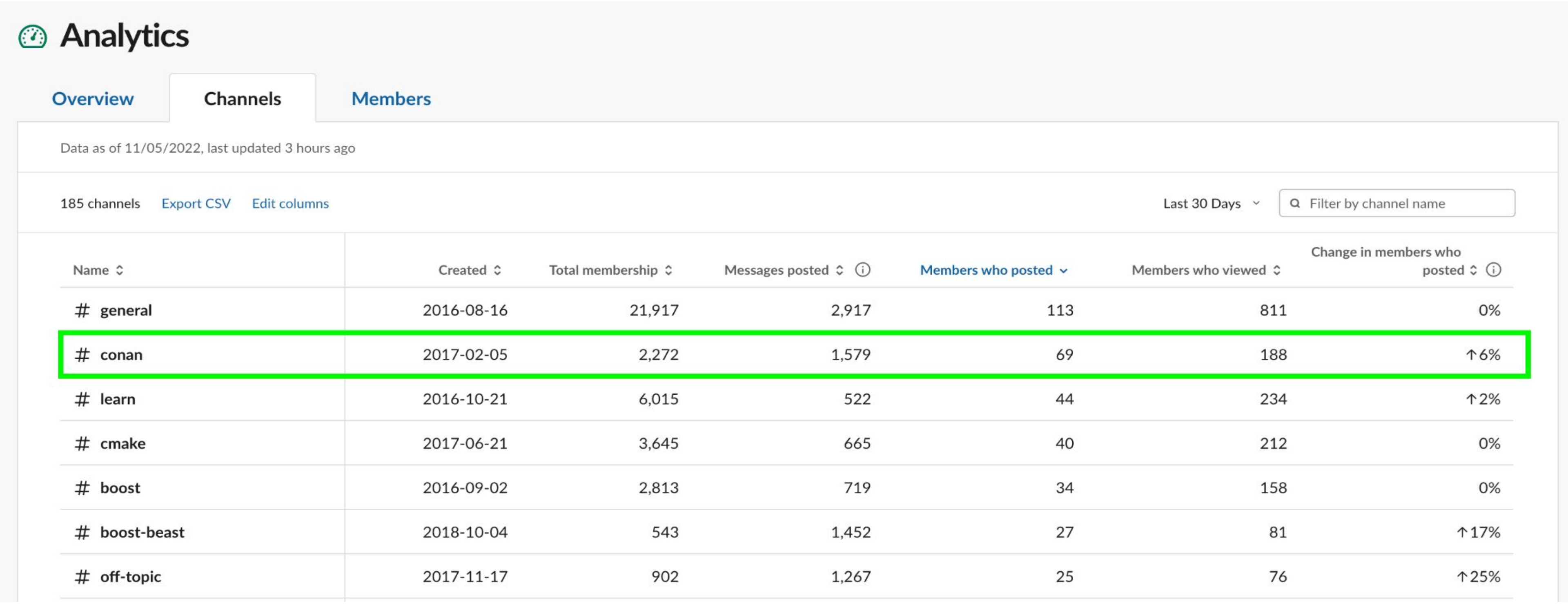
5 years, without breaking

60% new code, 20% backports

1.X ⇔ 2.0 compatible syntax subset

2.0

CppLang #conan slack



Analytics

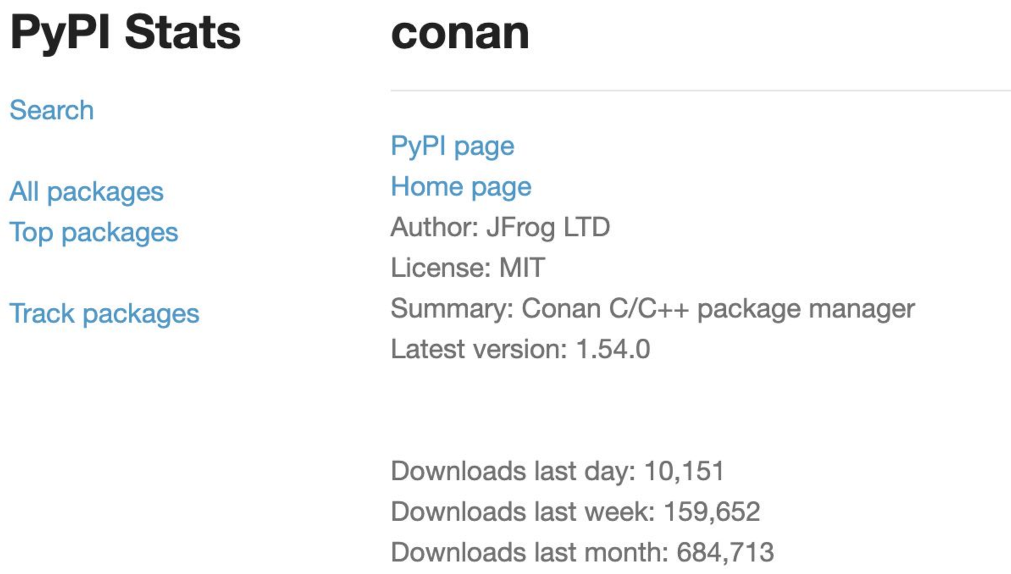
Data as of 11/05/2022, last updated 3 hours ago

185 channels Export CSV Edit columns Last 30 Days Filter by channel name

Name	Created	Total membership	Messages posted	Members who posted	Members who viewed	Change in members who posted
# general	2016-08-16	21,917	2,917	113	811	0%
# conan	2017-02-05	2,272	1,579	69	188	↑6%
# learn	2016-10-21	6,015	522	44	234	↑2%
# cmake	2017-06-21	3,645	665	40	212	0%
# boost	2016-09-02	2,813	719	34	158	0%
# boost-beast	2018-10-04	543	1,452	27	81	↑17%
# off-topic	2017-11-17	902	1,267	25	76	↑25%

PyPI downloads (Conan tool)

- 684K downloads/month from PyPI
- Designated as PyPI critical project (1% of most downloaded in whole PyPI)



PyPI Stats

Search

All packages
Top packages

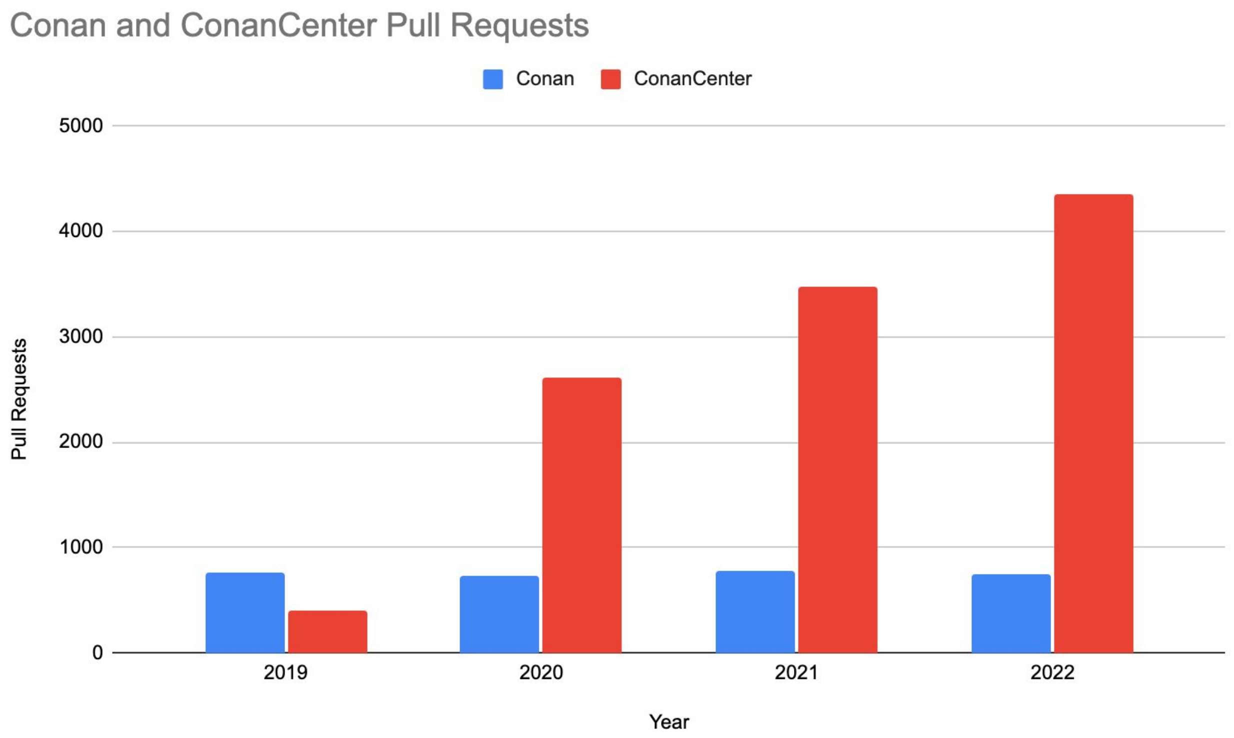
Track packages

conan

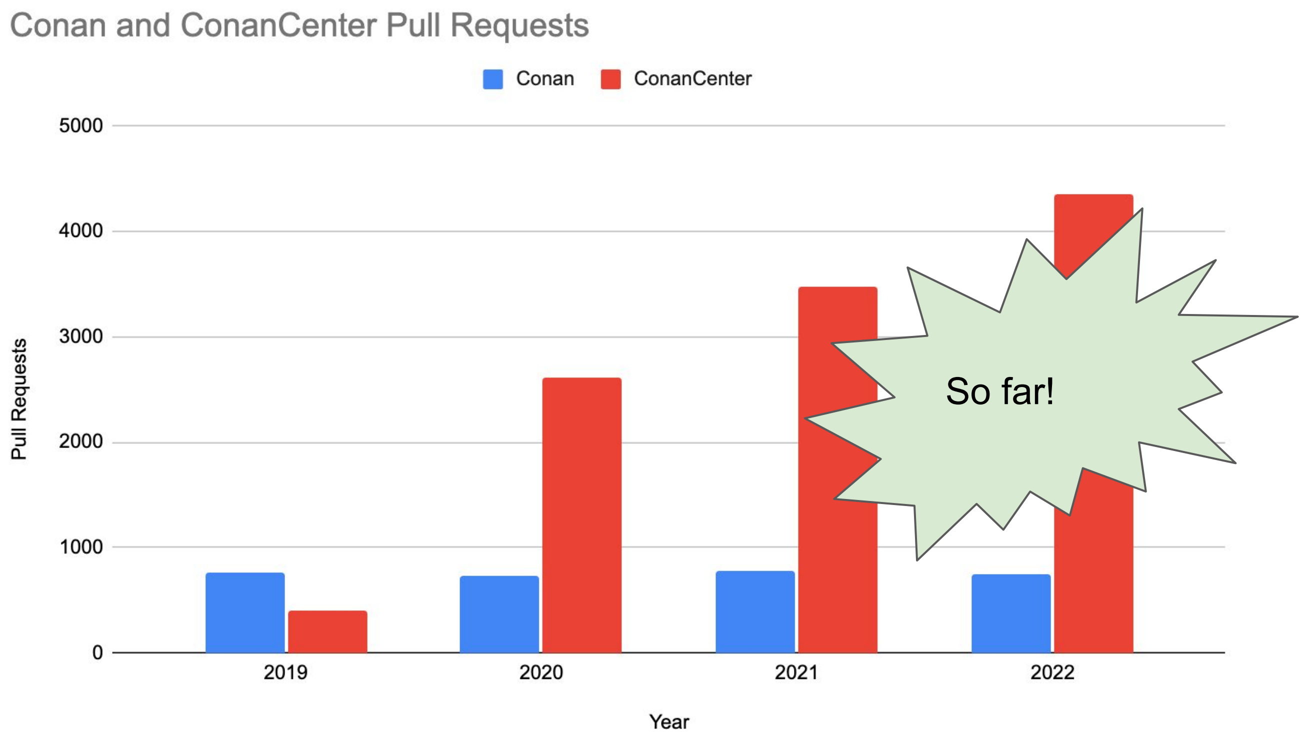
PyPI page
Home page
Author: JFrog LTD
License: MIT
Summary: Conan C/C++ package manager
Latest version: 1.54.0

Downloads last day: 10,151
Downloads last week: 159,652
Downloads last month: 684,713

Github PRs


Conan and ConanCenter Pull Requests
Conan
ConanCenter
5000
4000
3000
2000
1000
0
Pull Requests
2019
2020
2021
2022
Year

Github PRs

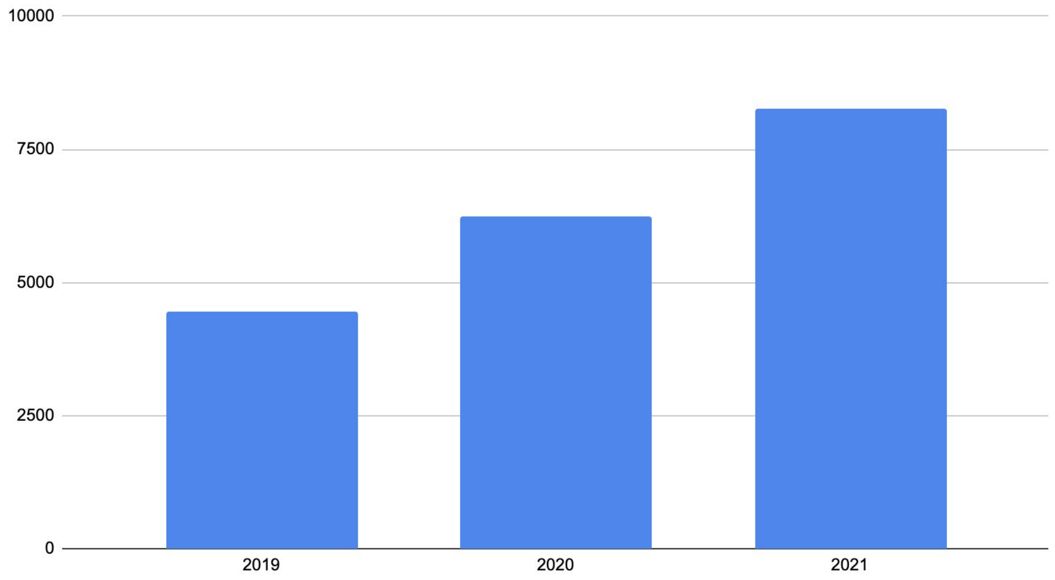

Conan and ConanCenter Pull Requests
Conan
ConanCenter
5000
4000
3000
2000
1000
0
Pull Requests
2019
2020
2021
2022
Year
So far!

Support

+2000 Github issues / year

100 hr/year user video calls

Direct support (slack, almost daily)

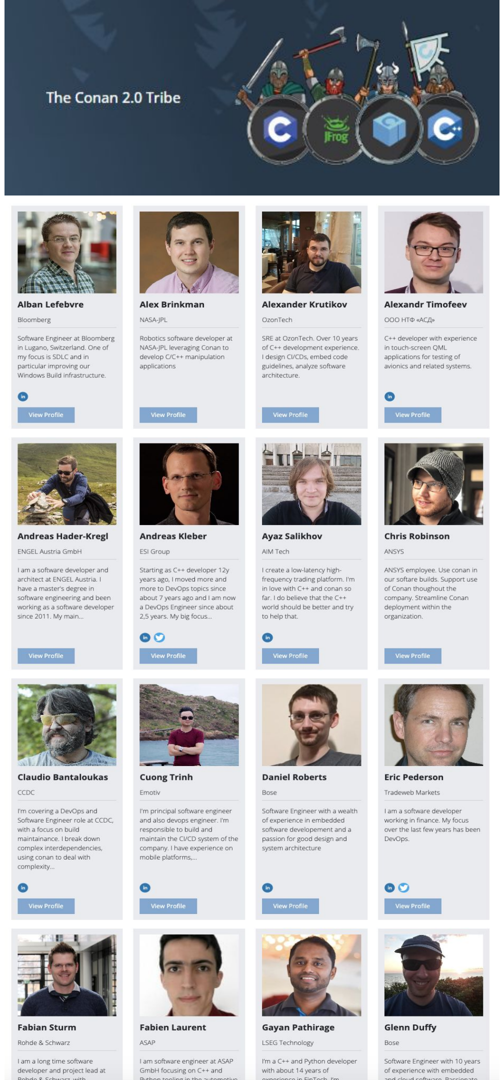


Artifactory servers running Conan in production and telemetry enabled (no firewalls)

Tribe 2.0 (conan.io/tribe.html)

- Bose
- TomTom
- Continental
- Nasa
- Apple
- Ansys
- Bloomberg
- Rohde & Schwarz
- Bosch

- ASAP
- Rti
- Zeiss
- Nasdaq
- Plex
- Keysight
- Datalogics
- VMWare
- … 50 more



Alban Lefebvre
Bloomberg
Software Engineer at Bloomberg in Lugano, Switzerland. One of my focus is SDLC and in particular improving our Windows Build infrastructure.
View Profile

Alex Brinkman
NASA-JPL
Robotics software developer at NASA-JPL leveraging Conan to develop C/C++ manipulation applications
View Profile

Alexander Krutikov
OzonTech
SRE at OzonTech. Over 10 years of C++ development experience. I design CI/CDs, embed code guidelines, analyze software architecture.
View Profile

Alexandr Timofeev
ООО НТФ «АСД»
C++ developer with experience in touch-screen QML applications for testing of avionics and related systems.
View Profile

Andreas Hader-Kregl
ENGEL Austria GmbH
I am a software developer and architect at ENGEL Austria. I have a master's degree in software engineering and been working as a software developer since 2011. My main...
View Profile

Andreas Kleber
ESI Group
Starting as C++ developer 12y years ago, I moved more and more to DevOps topics since about 7 years ago and I am now a DevOps Engineer since about 2,5 years. My big focus...
View Profile

Ayaz Salikhov
AIM Tech
I create a low-latency high-frequency trading platform. I'm in love with C++ and conan so far. I do believe that the C++ world should be better and try to help that.
View Profile

Chris Robinson
ANSYS
ANSYS employee. Use conan in our softare builds. Support use of Conan thoughout the company. Streamline Conan deployment within the organization.
View Profile

Claudio Bantaloukas
CCDC
I'm covering a DevOps and Software Engineer role at CCDC, with a focus on build maintainance. I break down complex interdependencies, using conan to deal with complexity...
View Profile

Cuong Trinh
Emotiv
I'm principal software engineer and also devops engineer. I'm responsible to build and maintain the CI/CD system of the company. I have experience on mobile platforms,...
View Profile

Daniel Roberts
Bose
Software Engineer with a wealth of experience in embedded software developement and a passion for good design and system architecture
View Profile

Eric Pederson
Tradeweb Markets
I am a software developer working in finance. My focus over the last few years has been DevOps.
View Profile

Fabian Sturm
Rohde & Schwarz
I am a long time software developer and project lead at

Fabien Laurent
ASAP
I am software engineer at ASAP GmbH focusing on C++ and

Gayan Pathirage
LSEG Technology
I'm a C++ and Python developer with about 14 years of

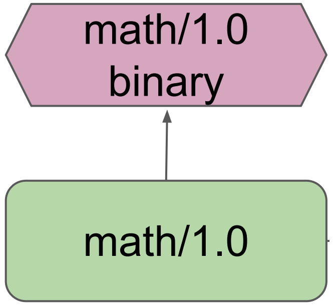
Glenn Duffy
Bose
Software Engineer with 10 years of experience with embedded

Overview

- 4 lessons:
  - Learning to fly
  - Building a dam
  - Dying of a thousand bites
  - Repeating yourself
- Conclusions

1. Learning to fly

Conanfile: A package “recipe”



```
$ git clone … math && cd math
$ conan create .
```

math/conanfile.py

```
from conan import ConanFile

class Math(ConanFile):
    name = "math"
    version = "1.0"

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

Conan 1.X dependency model: Transitive deps

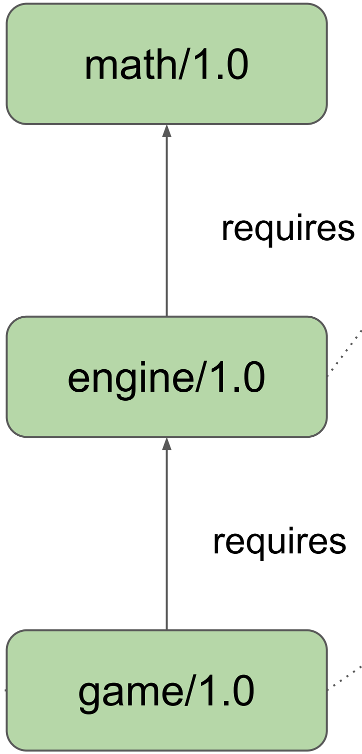
math-config.cmake

```
set_property(TARGET math::math …)
```

engine-config.cmake

```
set_property(TARGET engine::engine …)
```

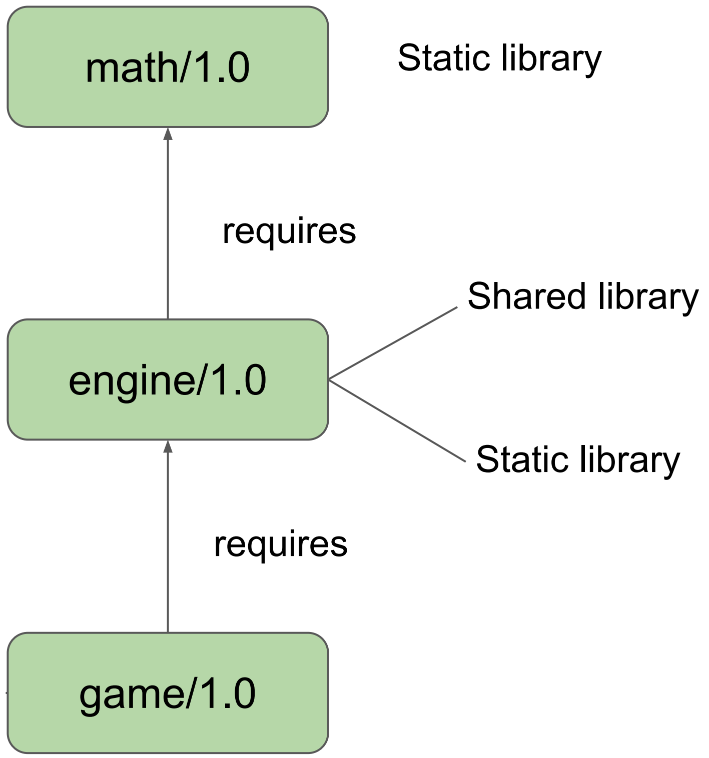
```
$ git clone … game && cd game
$ conan install .
$ cmake …
```



```python
from conan import ConanFile


class Engine(ConanFile):
    requires = "math/1.0"
```

```python
from conan import ConanFile


class Game(ConanFile):
    requires = "engine/1.0"
    generators = "CMakeDeps"
```

Conan 1.X dependency model: Transitive deps



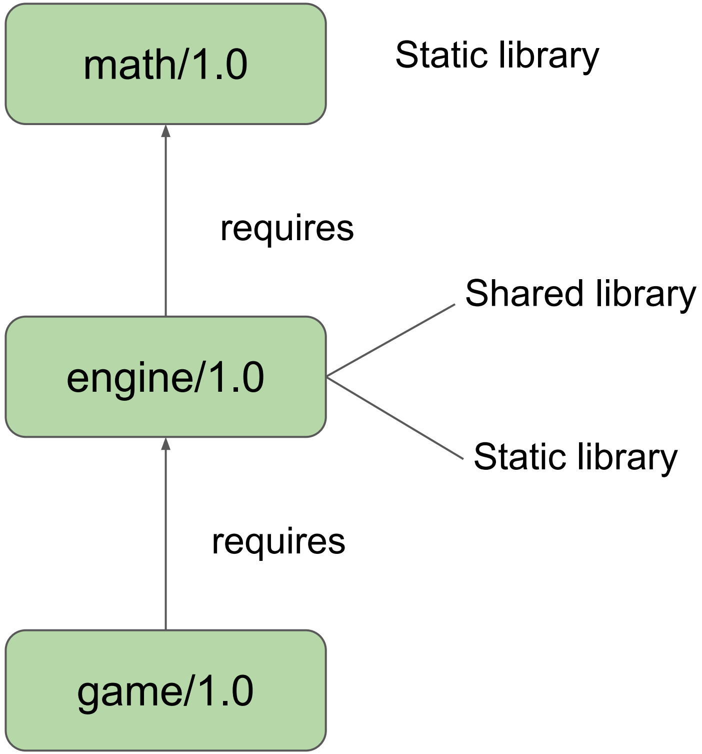
math-config.cmake

```
set_property(TARGET math::math ...)
```

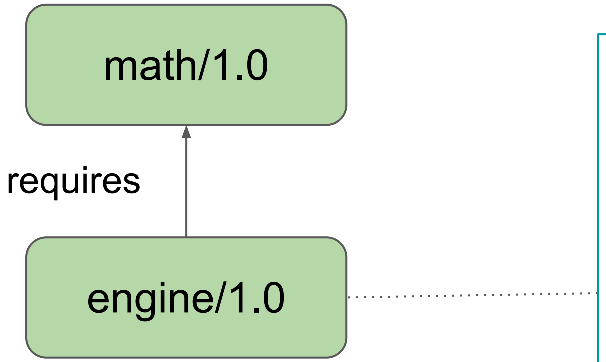
engine-config.cmake

```
set_property(TARGET engine::engine ...)
```

Learning to fly



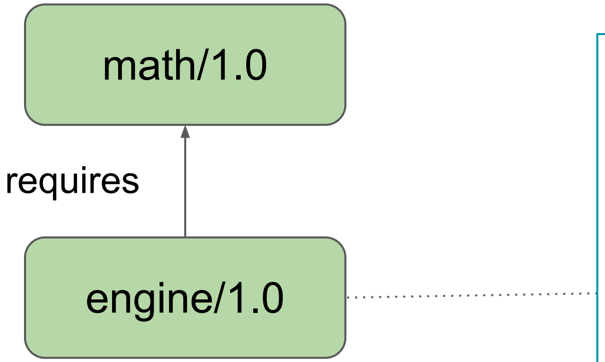
Conan 2.0 proposal



engine/conanfile.py

```python
from conan import ConanFile

class Engine(ConanFile):
    name = "engine"
    version = "1.0"

    def requirements(self):
        self.requires("math/1.0")
```

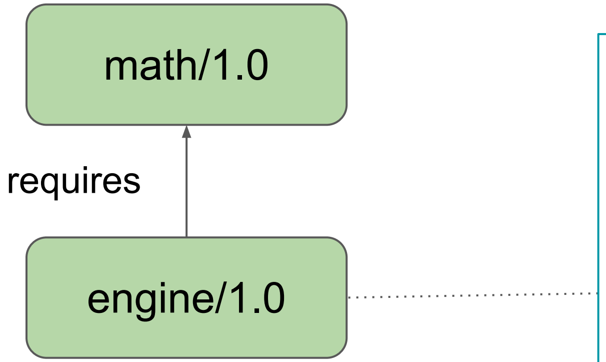
Conan 2.0 proposal: Requirement traits



engine/conanfile.py

```python
from conan import ConanFile


class Engine(ConanFile):
    name = "engine"
    version = "1.0"

    def requirements(self):
        self.requires("math/1.0",
                      headers=True, libs=True)
```

Conan 2.0 proposal: Requirement traits



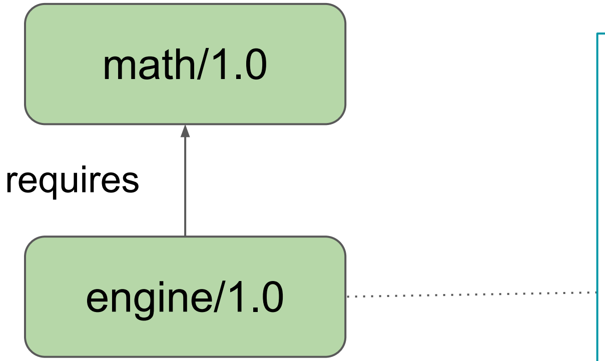
engine/conanfile.py

```python
from conan import ConanFile


class Engine(ConanFile):
    name = "engine"
    version = "1.0"

    def requirements(self):
        self.requires("math/1.0",
                      headers=True, libs=True)
```

math-config.cmake

```cmake
set_property(TARGET math::math PROPERTY INTERFACE_LINK_LIBRARIES ...)
set_property(TARGET math::math PROPERTY INTERFACE_INCLUDE_DIRECTORIES ...)
```

Conan 2.0 proposal: Requirement traits



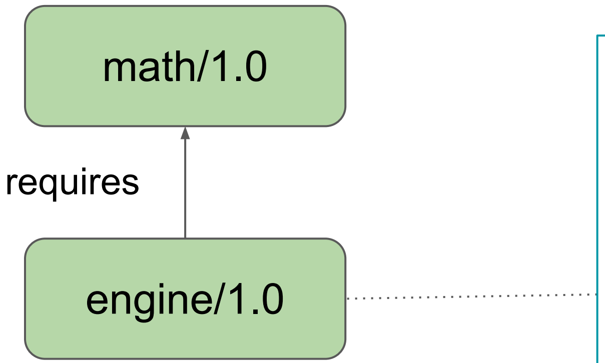
engine/conanfile.py

```
from conan import ConanFile


class Engine(ConanFile):
    name = "engine"
    version = "1.0"

    def requirements(self):
        self.requires("math/1.0",
                      headers=False, libs=True)
```

math-config.cmake

```
set_property(TARGET math::math PROPERTY INTERFACE_LINK_LIBRARIES ...)
~~set_property(TARGET math::math PROPERTY INTERFACE_INCLUDE_DIRECTORIES~~ ...)
```

Conan 2.0 proposal: Requirement traits



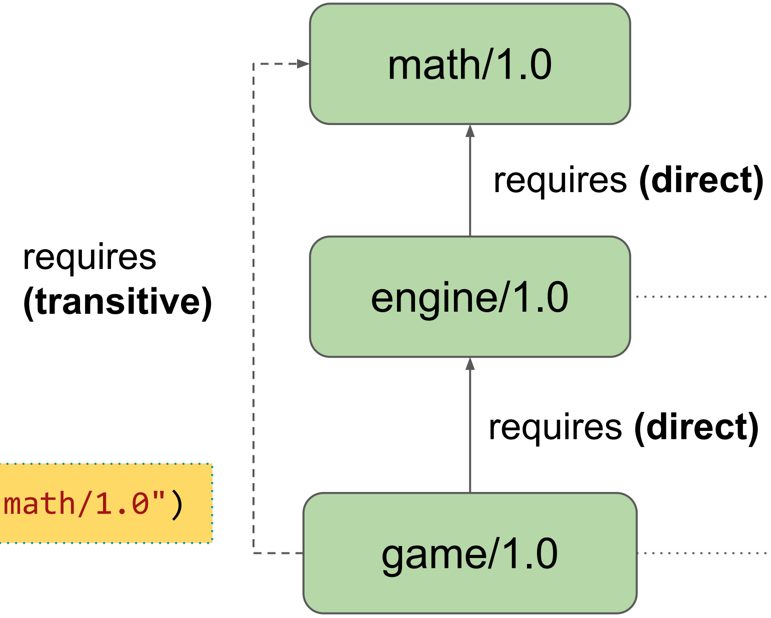
engine/conanfile.py

```
from conan import ConanFile


class Engine(ConanFile):
    name = "engine"
    version = "1.0"

    def requirements(self):
        self.requires("math/1.0",
                                    headers=True, libs=False)
```

math-config.cmake

```
~~set_property(TARGET math::math PROPERTY INTERFACE_LINK_LIBRARIES ...)~~
set_property(TARGET math::math PROPERTY INTERFACE_INCLUDE_DIRECTORIES ...)
```

Conan 2.0 proposal: Direct vs. transitive dependencies

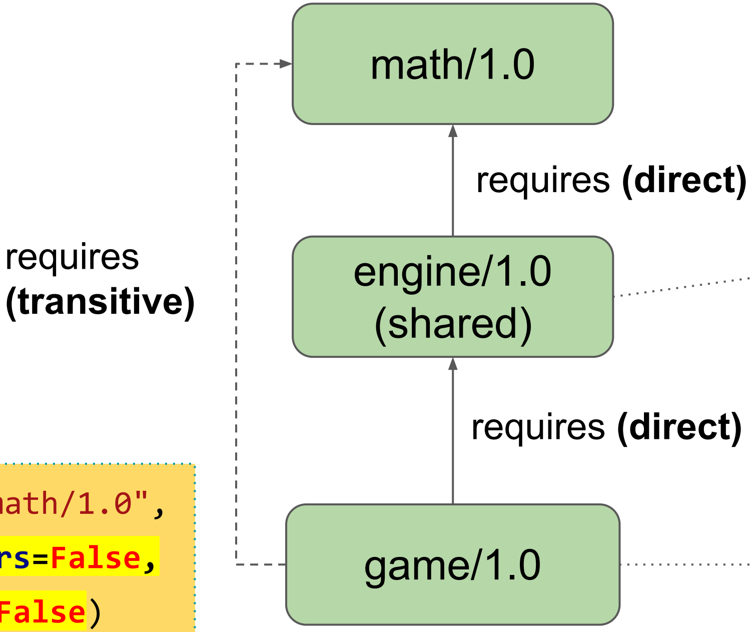


```
self.requires("math/1.0")
```

```
class Engine(ConanFile):
    def requirements(self):
        self.requires("math/1.0")
```

```
class Game(ConanFile):
    def requirements(self):
        self.requires("engine/1.0")
```

Linkage requirements propagation



```python
class Engine(ConanFile):
    def requirements(self):
        self.requires("math/1.0",
            headers=True, libs=True,
            transitive_libs=False)
```

```python
class Game(ConanFile):
    def requirements(self):
        self.requires("engine/1.0",
            headers=True, libs=True)
```

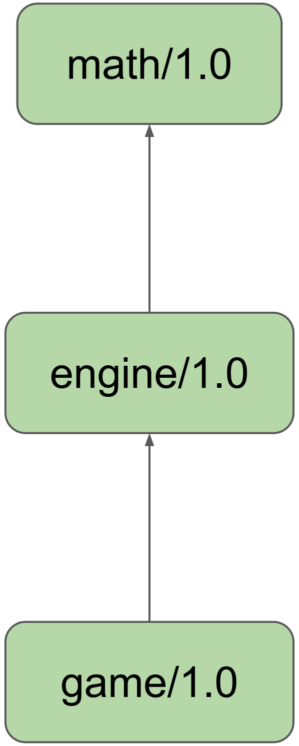
```python
self.requires("math/1.0",
              headers=False,
              libs=False)
```

math-config.cmake

~~set_property(TARGET math::math PROPERTY INTERFACE_LINK_LIBRARIES ...)~~

~~set_property(TARGET math::math PROPERTY INTERFACE_INCLUDE_DIRECTORIES ...)~~

Package Types



math/conanfile.py

```
class Math(ConanFile):
    name = "math"
    version = "1.0"
    package_type = "static-library"
    # OR options = {"shared": [True, False]}
```

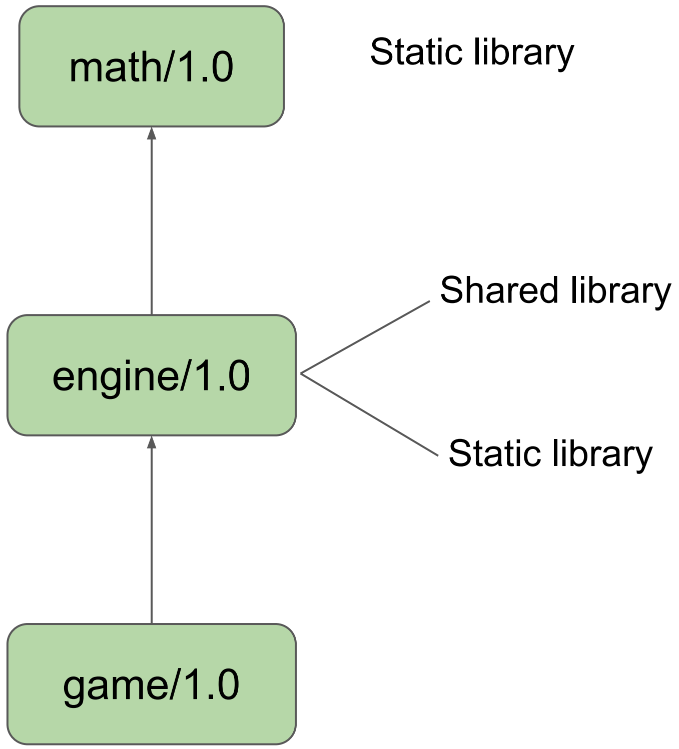
engine/conanfile.py

```
class Engine(ConanFile):
  package_type = "shared-library"
  # OR options = {"shared": [True, False]}
  def requirements(self):
    self.requires("math/1.0")
```

game/conanfile.py

```
class Game(ConanFile):
  package_type = "application"
  def requirements(self):
    self.requires("engine/1.0")
```

Demo


math/1.0
Static library
engine/1.0
Shared library
Static library
game/1.0

Dependency graph 2.0

- Correct linkage requirements
- Correct header visibility
- Possible hidden/private dependencies
- and many more (ACCU 2022)

Among different build systems!

Compatible “requires” syntax with 1.X

2. Building a dam

App build & runs: great job!

Extremely opinionated ecosystem

They: I want the libs from Conan dependencies in my project folder

Us: No need for it, you can use the libs from the cache

They: But the dependencies should be in the project

Us: Not really, many other package managers Maven, pip, do not put dependencies in your project

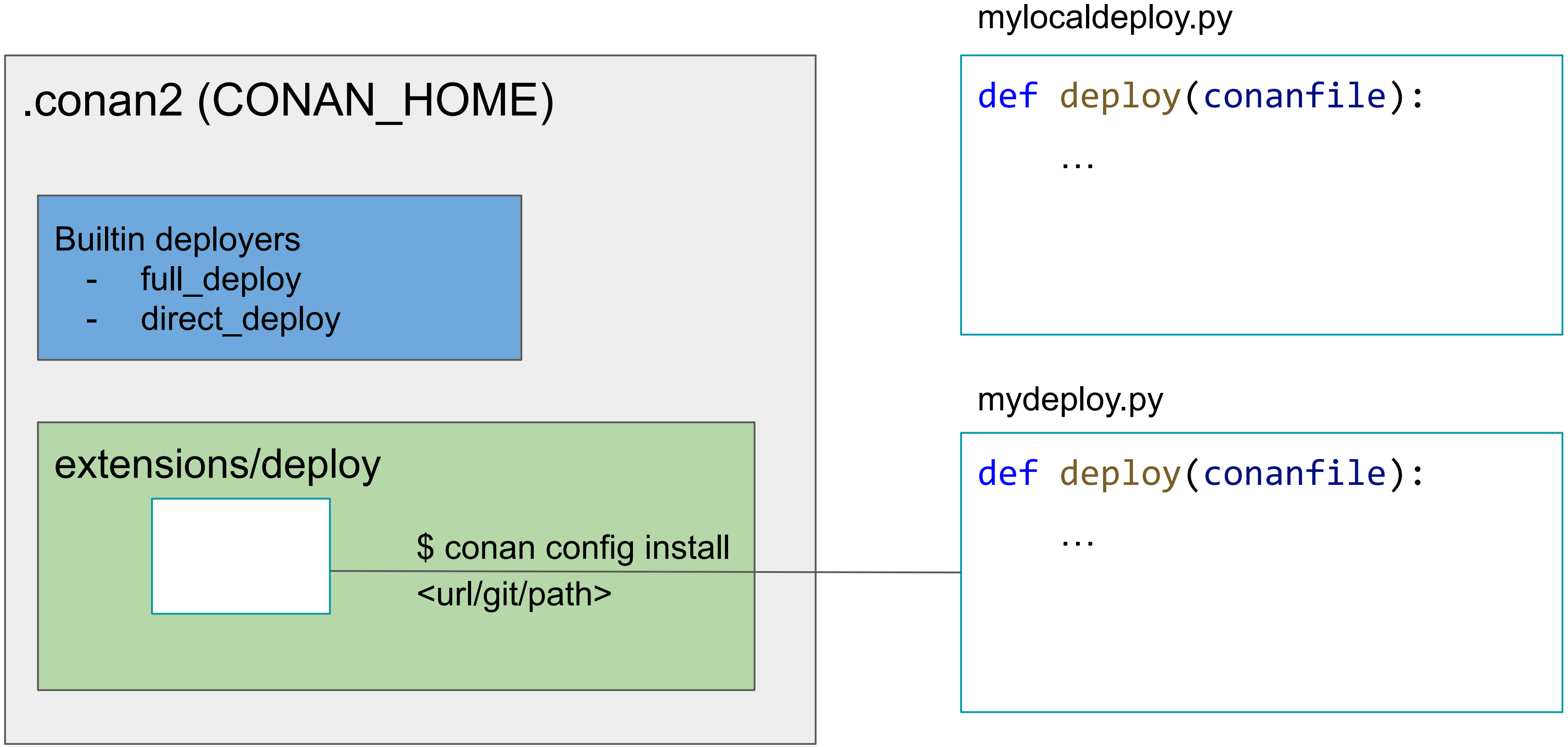
They: But it is easy, why don’t you just put the dependencies libs in my project folder

Us: It is easy that they will conflict, different versions of the same, or different binaries, no metadata, no synchronization, more space in disk

…

They: I want the libs in my project folder, they should be there

Deployers



mylocaldeploy.py

```
def deploy(conanfile):
    ...
```

mydeploy.py

```
def deploy(conanfile):
    ...
```

Demo

Deployers

- Flexible way to extract artifacts from cache
- Automate post-conan tasks
- Not in recipes, scale
- User customizable, “conan config install” installable

3. Dying of a thousand bites

Plugins

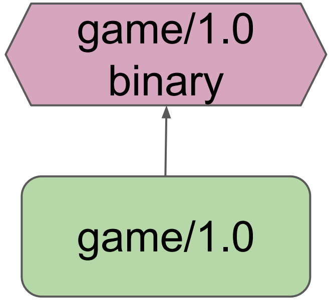
The solution - empower users to do it themselves!

Provide a framework for users to build solutions tailored to their needs with mechanisms that give them controlled management.

- Profile
- Command Wrapper
- Package Signing – Demo

Profile Plugin

Let’s build our game for the local developer’s system

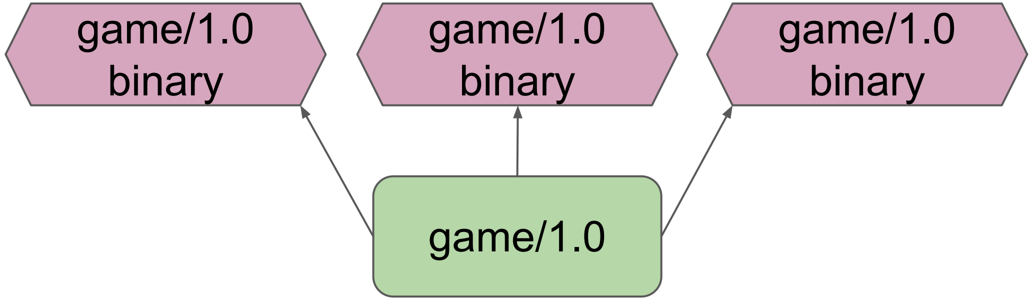


profiles/linux-gcc

```
[settings]
os=Linux
arch=x86_64
build_type=Release
compiler=gcc
compiler.cppstd=gnu20
compiler.libcxx=stdlib++11
compiler.version=8
```

Profile Templates (1.x)

What if we need to build, test and ship for multiple versions?



profiles/linux-gcc-#

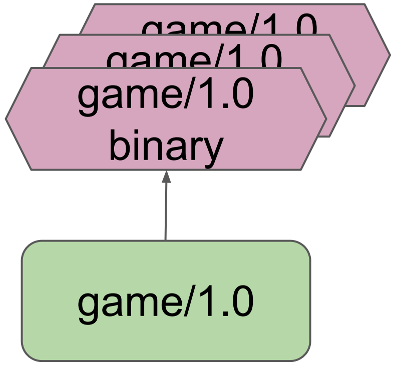
```
[settings]
os=Linux
arch=x86_64
build_type=Release
compiler=gcc
compiler.cppstd=gnu20
compiler.libcxx=stdlib++11
compiler.version=8, 10, 12
```

Profile Templates (1.x)

Given just one profile we can now build 8+ combinations of the game binary

profiles/linux-gcc

```
[settings]
os=Linux
arch=x86_64
build_type= {{ os.getenv("MY_BUILD_TYPE") }}
compiler=gcc
compiler.cppstd=gnu20
compiler.libcxx=stdlib++11
compiler.version= {{ os.getenv("MY_GCC_VER") }}
```



Profile Templates (1.x)

profiles/linux-gcc

```
[settings]
compiler=gcc
compiler.cppstd=gnu20
compiler.libcxx=stdlib++11
compiler.version={{ os.getenv("MY_GCC_VER") }}
```

Dev’s enviroment

```
$ export MY_GCC_VER=6
```

profiles/linux-gcc

```
[settings]
compiler=gcc
compiler.cppstd=gnu20
compiler.libcxx=stdlib++11
compiler.version=6
```

Profile Plugin

How can we ensure that the profiles being used are valid settings?

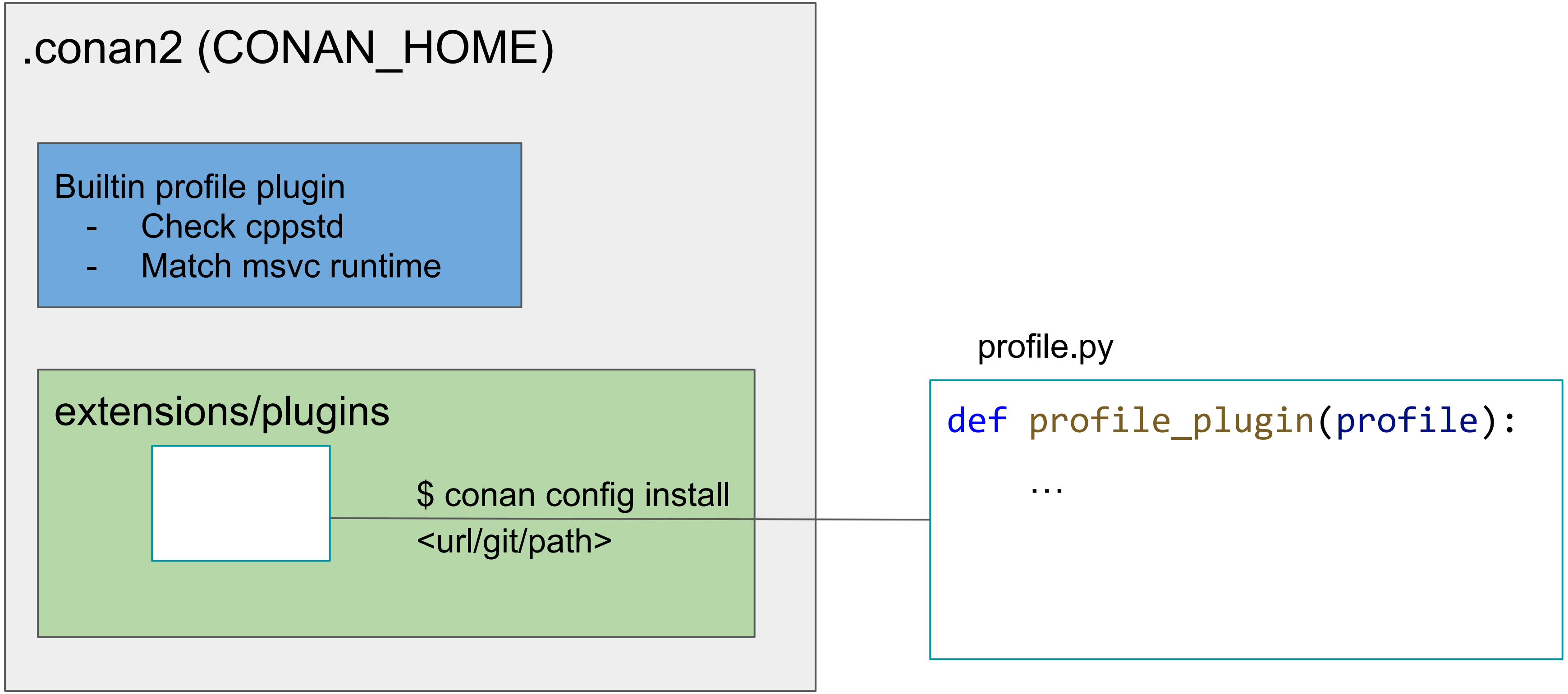
gcc-6 with c++20 ([which was introduced in gcc-8]())

apple-clang 12 with c++23

profiles/linux-gcc-5

```
[settings]
os=Linux
arch=x86_64
build_type=Release
compiler=gcc
compiler.cppstd=gnu20
compiler.libcxx=stdlib++11
compiler.version=6
```

Profile Plugin



```python
def profile_plugin(profile):
    ...
```

Profile Plugin

Check ``cppstd`` ensure the settings being used exist for the version of the compiler.

profiles/macos-x86-ac14-23

```
[settings]
os=Macos
arch=x86_64
build_type=Release
compiler=apple-clang
compiler.cppstd=23
compiler.libcxx=libc++
compiler.version=14
```

Sample output

```
$ conan create game -s compiler.version=12 -s compiler.cppstd=23
ERROR: The provided compiler.cppstd=23 requires at least apple-clang>=13 but version 12
provided
```

Profile Plugin

Picking MSVC Runtime Optimization

- Depending on compilation optimization
- Use the matching runtime /MT, /MTd, etc..

Changing from Debug to Release will be applied through out.

Profile Plugins

There’s two plugins that are included with Conan 2.0

- Check if ``cppstd`` supported by the compiler.
    - This was hardcoded in 1.x
- Visual studio runtime usually match the build type
    - Can now be set by profile, so ``-s build_type=Debug **-s compiler.runtime=Debug**``
    - You can disable this rule, but it's available for 2.0 migration

User defined and extensible can be tailor to enforce workplace or project specifics conventions, contratins, or compliance.

Command Wrapper

Allows you directly manipulate the ``self.run`` calls with extra arguments or variables.

extensions/plugins/cmd_wrapper.py

```
def cmd_wrapper(cmd):
   if cmd.starts_with("cmake"):
       return "CMAKE_CXX_COMPILER_LAUNCHER=ccache {}"'.format(cmd)
   return cmd
```

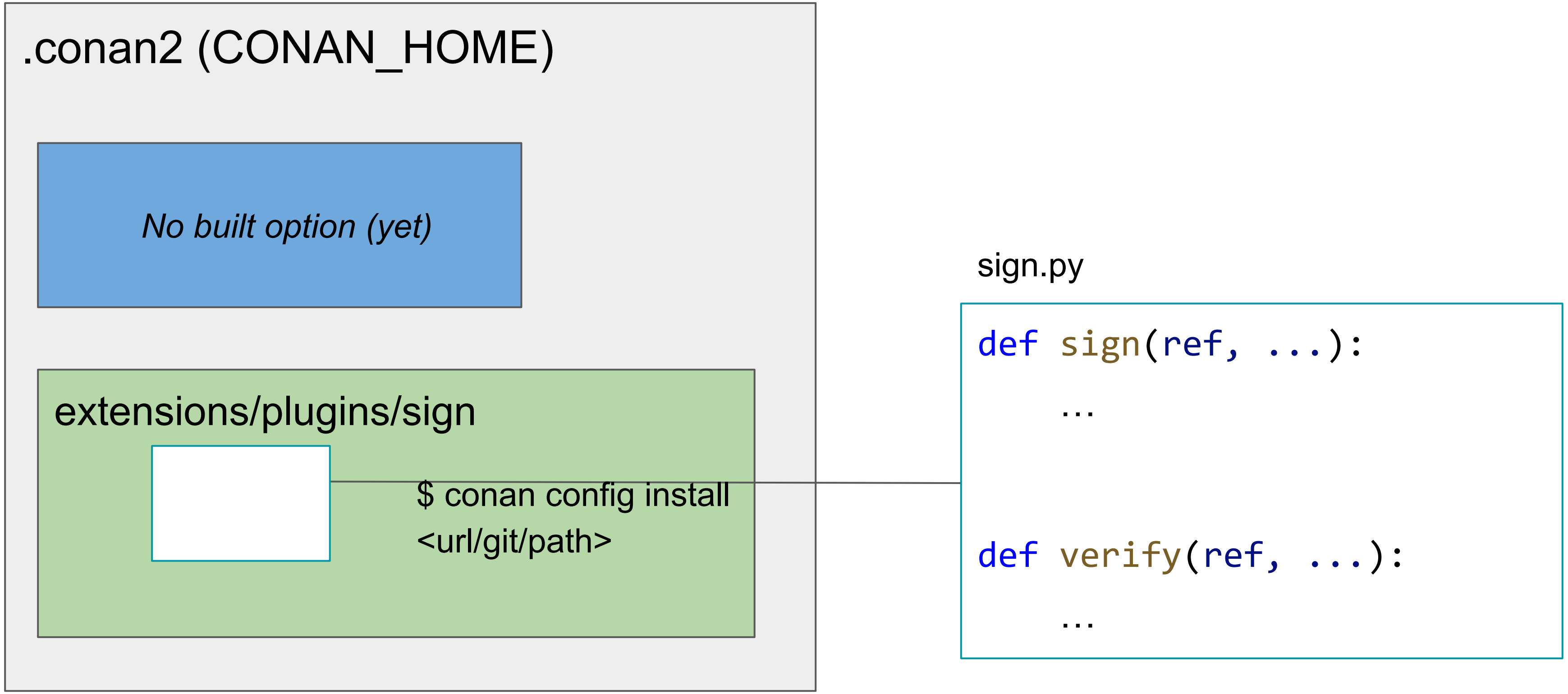
For example we can intercept all the calls to CMake and make sure the variables for compiler launcher is set this way we can have ccache being used to speed up build times

Package Signing

Absolutely critical to addressing supply chain security. The one feature offers the most room for innovation within the C++ Open Source ecosystem.

Completely extensible to allow existing solutions are new external integrations to be developed or incorporated.

Packaging Signing



```
$ conan config install <url/git/path>
```

```
def sign(ref, ...):
    ...


def verify(ref, ...):
    ...
```

Package Signing

Takes when talking to a remote (i.e not invoked when creating packages locally)

- ``sign`` place when uploading recipes
- ``verify`` takes place during install

These two methods will able to compute signatures and read/write them to a special “signing data folder” in the cache to be reused.

Demo

Plugins

This will put users in control and that’s not to mention custom commands or the python API which I did not share today.

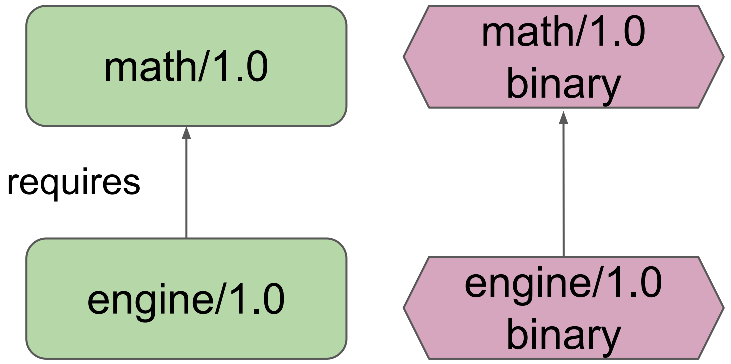
You’ll need to watch Diego’s CppCon for that.

Stay tuned we have an exciting news about integrations - conan_io on twitter or subscribe to the newsletter.

4. Repeating yourself

Binary Compatibility

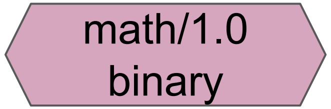
What exactly does this mean? We’ll depends who you ask to let me explain the perspective of Conan and how it images packages



Compatible if I change compiler version???

Binary Compatibility

Binary packages each have unique ID regardless of compatibility

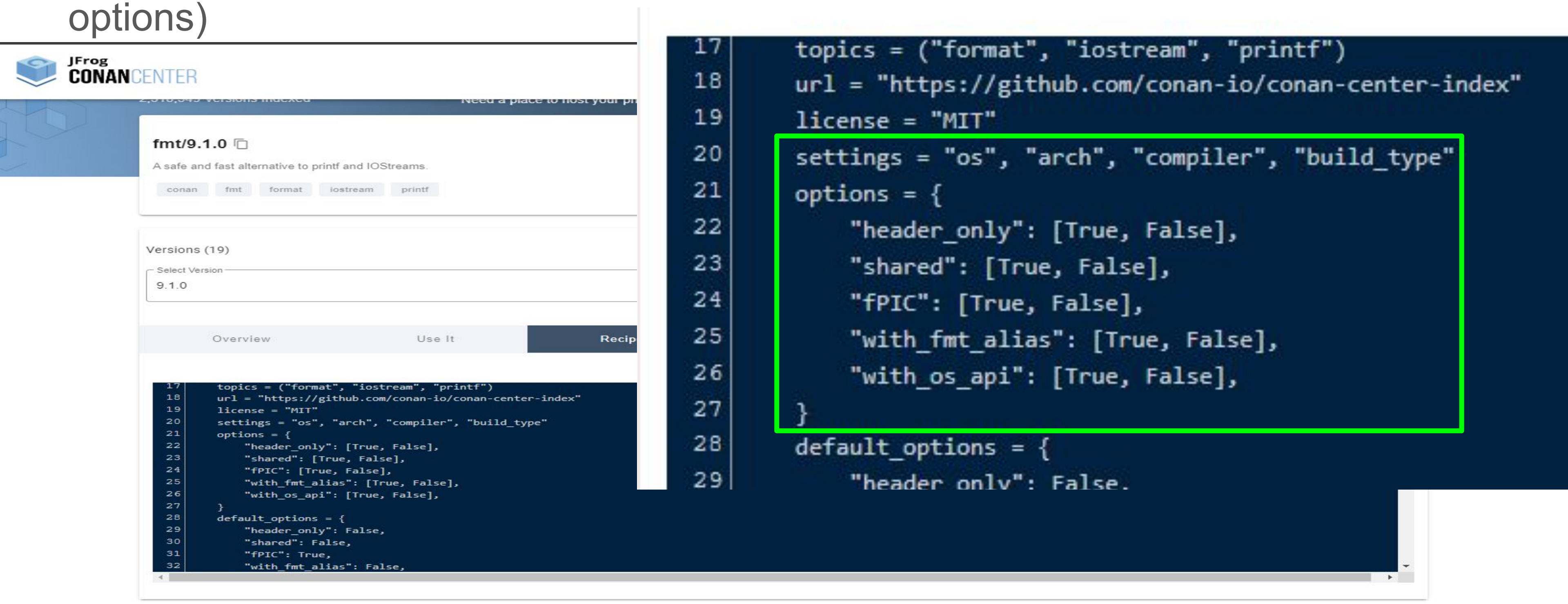


Package_ID: 6af9cc7cb931c5ad942174fd7838eb655717c709

Different configurations – match exactly the same settings (must be compatible) – except when it’s not…

Binary Compatibility

Packages IDs are computer from the binary model of the recipe (settings and options)



fmt/9.1.0

A safe and fast alternative to printf and IOStreams.

Versions (19)

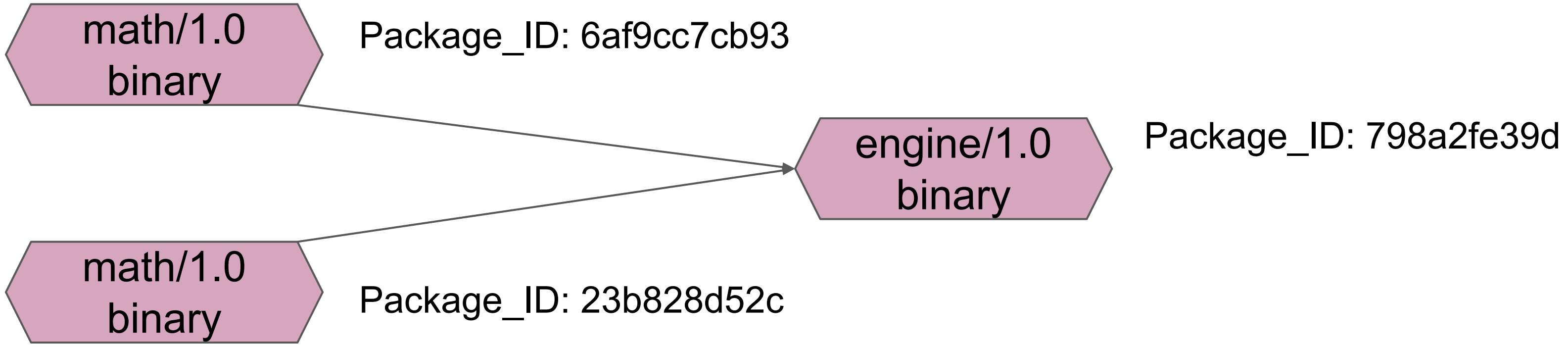
Select Version: 9.1.0

```
    topics = ("format", "iostream", "printf")
    url = "https://github.com/conan-io/conan-center-index"
    license = "MIT"
    settings = "os", "arch", "compiler", "build_type"
    options = {
        "header_only": [True, False],
        "shared": [True, False],
        "fPIC": [True, False],
        "with_fmt_alias": [True, False],
        "with_os_api": [True, False],
    }
    default_options = {
        "header_only": False,
        "shared": False,
        "fPIC": True,
        "with_fmt_alias": False,
```

```
    topics = ("format", "iostream", "printf")
    url = "https://github.com/conan-io/conan-center-index"
    license = "MIT"
    settings = "os", "arch", "compiler", "build_type"
    options = {
        "header_only": [True, False],
        "shared": [True, False],
        "fPIC": [True, False],
        "with_fmt_alias": [True, False],
        "with_os_api": [True, False],
    }
    default_options = {
        "header_only": False,
```

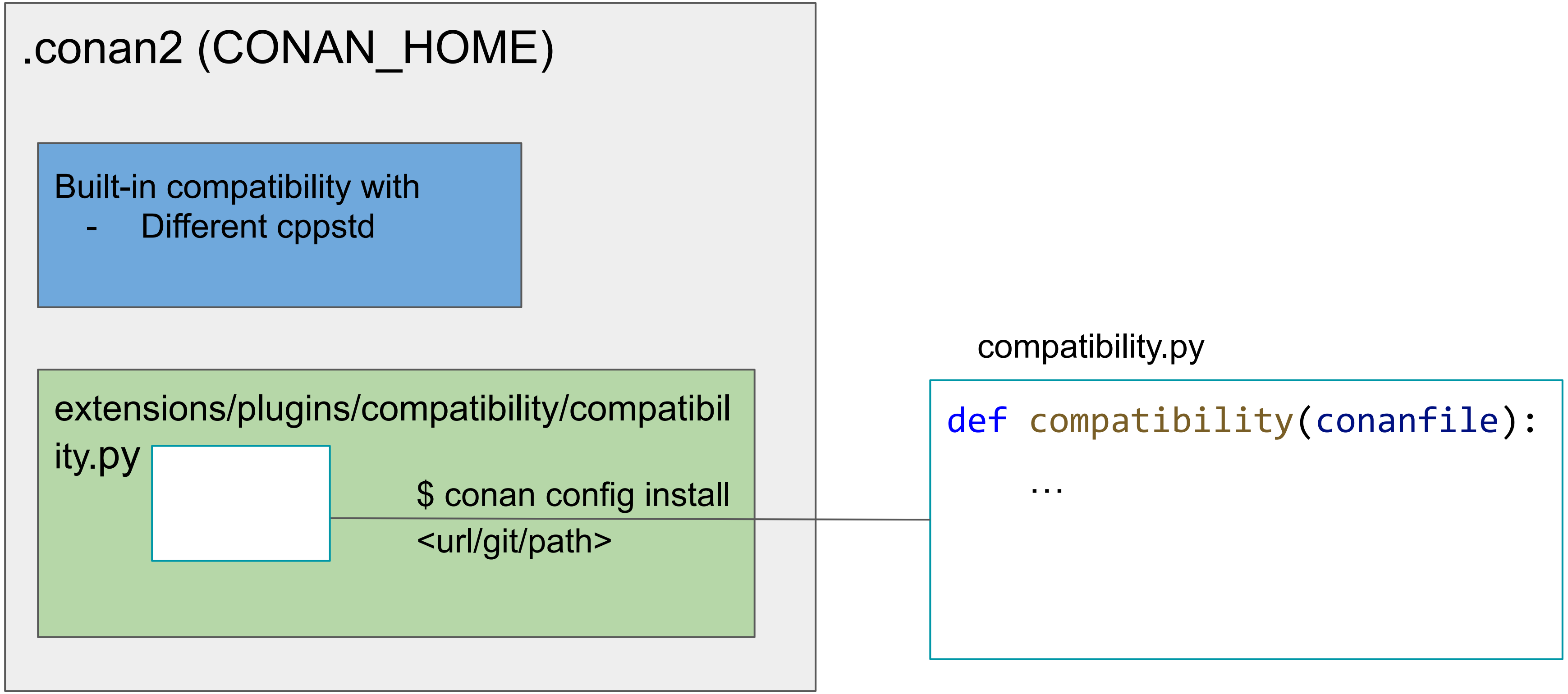
Binary Compatibility

So compatibility in Conan means different package IDs and be interchangeable and still result in a valid final binary



Different inputs – same output

Compatibility Plugin


.conan2 (CONAN_HOME)
Built-in compatibility with
- Different cppstd
extensions/plugins/compatibility/compatibility.py
compatibility.py

```
$ conan config install <url/git/path>
```

```python
def compatibility(conanfile):
    ...
```

Demo

Picking a lower ``cppstd`` then in our settings

Let’s build math and engine with cppstd 14

Let’s build our game with cppstd 17 is should find compatible packages for cppstd 14 for the two dependencies

– > Starts with math cppstd 17 if not found it will look for 14 (not just mixing)
There’s a defined priority queue

By default it’s a deterministic list — you can change this and write your own! Another build type or compiler version it’s your choice!

Conclusions

New graph

New plugin extensions

New deployers

New binary compatibility

Multi-revision cache

package_id

Lockfiles

New configuration and environment

Package immutability optimizations

… and many more

https://docs.conan.io/en/2.0/whatsnew.html

Conclusion

```
pip install conan==2.0-beta.5
```

https://conan.io