# C++ Modernization

## From Monolith Monorepo to Scalable Cloud Microservices

November 2022

Benoit Lavallee

Team lead, Coveo

## Agenda

1. Overview of C++@Coveo
2. Switch from monolith on-premises app to multiple microservices with “every day” deployment
3. Build automation
4. Package manager
5. Examples
6. Questions

## 1. Overview of C++@Coveo

# A little bit of context

- 2000 : First lines of code
- 2002 - 2012  : On-premises (Windows monolith application)
- 2012 - 2014  : Cloud version 1 (Windows monolith application)
- 2014 - today : Cloud version 2 (Linux “microservices”)

# C++ Source Code at Coveo

| Project | File Count | Blank Lines | Comment Lines | Header Lines[1] | Source (.cpp) Lines[1] |
|---|---|---|---|---|---|
| **Basic Lib** | 1 587 | 54 809 | 130 366 | 79 367 | 298 940 |
| **Comm. Lib** | 1 026 | 39 210 | 99 137 | 48 971 | 120 343 |
| Subtotal | **2 613** | **94 019** | **229 503** | **128 338** | **419 283** |
| | | | | | |
| **Common Lib** | 640 | 12 946 | 30 284 | 15 884 | 38 009 |
| **Pipeline** | 220 | 7 176 | 13 254 | 6 608 | 24 176 |
| **Converters** | 446 | 14 749 | 36 304 | 24 776 | 70 307 |
| **Index** | 1 340 | 64 965 | 111 874 | 73 689 | 271 992 |
| Subtotal | **2 646** | **99 836** | **191 716** | **120 957** | **404 484** |
| | | | | | |
| **Total** | **5 259** | **193 855** | **421 219** | **249 295** | **823 767** |

Utility code.
Like what is found in the stl, boost, tbb, curl, …

Lot of legacy code. A significant part of it could be replaced with the stl, boost, …

Main code.
This contains what we sell to our clients.

Depends on legacy utility code.
Contains older code that is not well adapted to modern computers.

(1) Total number of source lines in headers and .cpp files counted using cloc.

## 2. Switch from monolith on-premises app to microservices with “every day” deployment

## Phase 1

- Multiplatform (Windows only → Windows & Linux)
  - Use msbuild even in Linux with mono
    - msbuild calls cmake commands on Linux
- Multiple repositories (step 1)
  - Extract basic classes and some big logical entities
- Dockerization
  - Create a docker for each microservices but in the same repo
- All services were deployed at the same time

## Phase 2

- Tests must be reliable
- Use a real cross-platform build automation
  - CMake was selected for Windows and Linux
- Use a package manager
  - vcpkg was selected
- Multi repos (step 2)
  - Create a repository for each microservices
- Trunk based development … when it’s possible
  - Each services can be deploy anytime

## 3. Build automation

CMake

## CMake >= 3

- Pros
  - CTest, CPackage
  - Supports an extensive list of compilers
  - Support in
- Cons
  - The scripting language

## 4. Package manager

vcpkg

## Vcpkg

- Pros
  - CMake integration
  - Large number of packages available
  - Maintained by Microsoft
- Cons
  - Relatively new

## 5. Examples

## Examples

- Arm64
  - In **2 weeks** all C++ is running (including Unit tests)
- Fix vulnerability in openssl 3
  - All instances (**~3000**) are updated **48h** after fix available
- New OS version
  - Upgrading to a new Ubuntu LTS is now about a **few weeks**
    - including the time to build with the new compiler version

## 6. Questions?