algolia

AI SEARCH

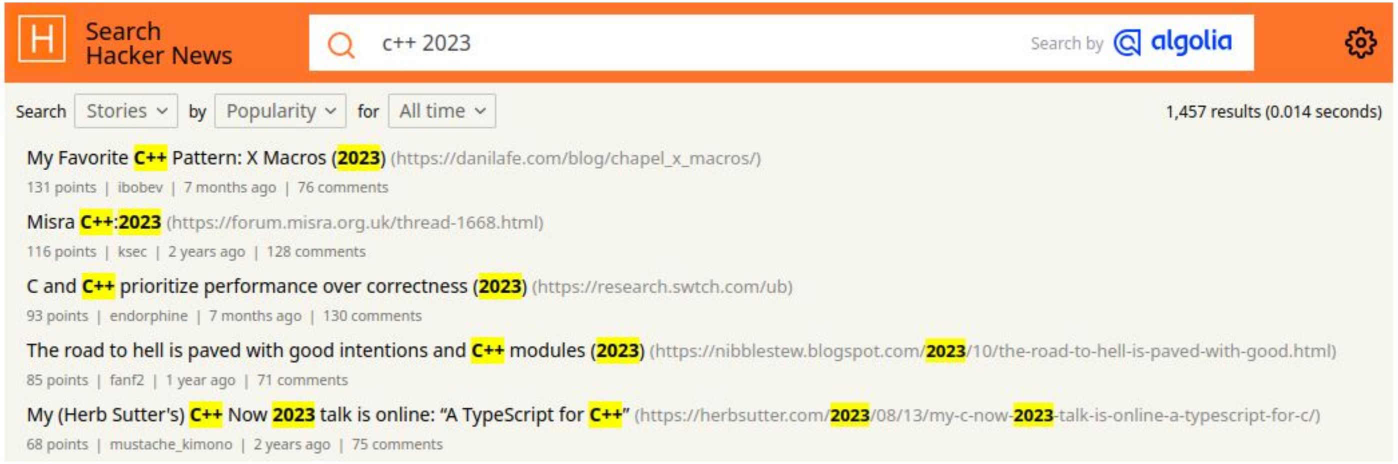
sccache + conan

Réparer les fondations sans tout casser

Bertrand Le Mée

Algolia - un moteur de recherche en SaaS

- API pour indexer et retrouver des documents
- Résultat final: une barre de recherche sur un site client
- La pertinence, un sujet sans fin: recherche sémantique, RAG...



1,457 results (0.014 seconds)

My Favorite C++ Pattern: X Macros (2023) (https://danilafe.com/blog/chapel_x_macros/)
131 points | ibobev | 7 months ago | 76 comments

Misra C++:2023 (https://forum.misra.org.uk/thread-1668.html)
116 points | ksec | 2 years ago | 128 comments

C and C++ prioritize performance over correctness (2023) (https://research.swtch.com/ub)
93 points | endorphine | 7 months ago | 130 comments

The road to hell is paved with good intentions and C++ modules (2023) (https://nibblestew.blogspot.com/2023/10/the-road-to-hell-is-paved-with-good.html)
85 points | fanf2 | 1 year ago | 71 comments

My (Herb Sutter's) C++ Now 2023 talk is online: "A TypeScript for C++" (https://herbsutter.com/2023/08/13/my-c-now-2023-talk-is-online-a-typescript-for-c/)
68 points | mustache_kimono | 2 years ago | 75 comments

Un moteur à grande échelle

17k utilisateurs

100GiB par app

20ms par query

3k+ serveurs

99.999% uptime

1.75B queries

200k+ QPS

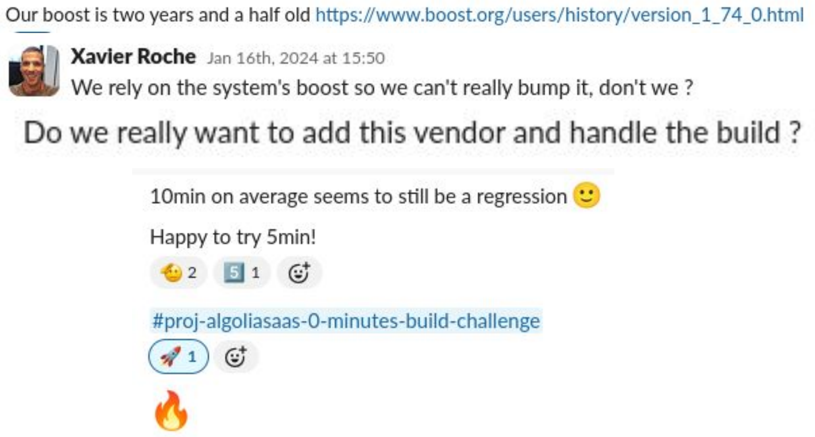
Un moteur vieillissant

- Premier commit en 2013
- ~580 kLOC de C++
- 36 libs en dépendances

Au cœur de toutes les fonctionnalités

Budget dette technique:
20% du temps de dev

Dette technique



Our boost is two years and a half old https://www.boost.org/users/history/version_1_74_0.html

Xavier Roche Jan 16th, 2024 at 15:50
We rely on the system's boost so we can't really bump it, don't we ?

Do we really want to add this vendor and handle the build ?

10min on average seems to still be a regression 🙂

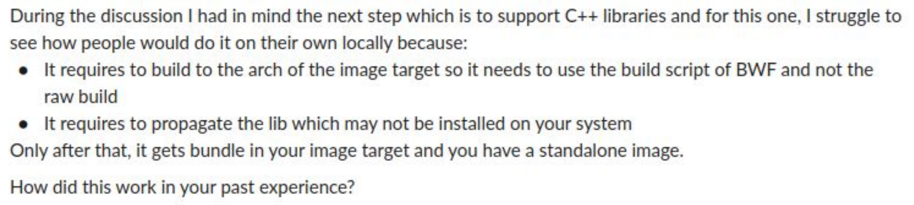
Happy to try 5min!

🫡 2 5️⃣ 1

#proj-algoliasaas-0-minutes-build-challenge

🚀 1

🔥

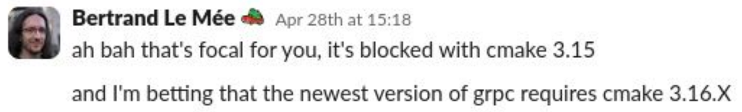


During the discussion I had in mind the next step which is to support C++ libraries and for this one, I struggle to see how people would do it on their own locally because:

- It requires to build to the arch of the image target so it needs to use the build script of BWF and not the raw build
- It requires to propagate the lib which may not be installed on your system

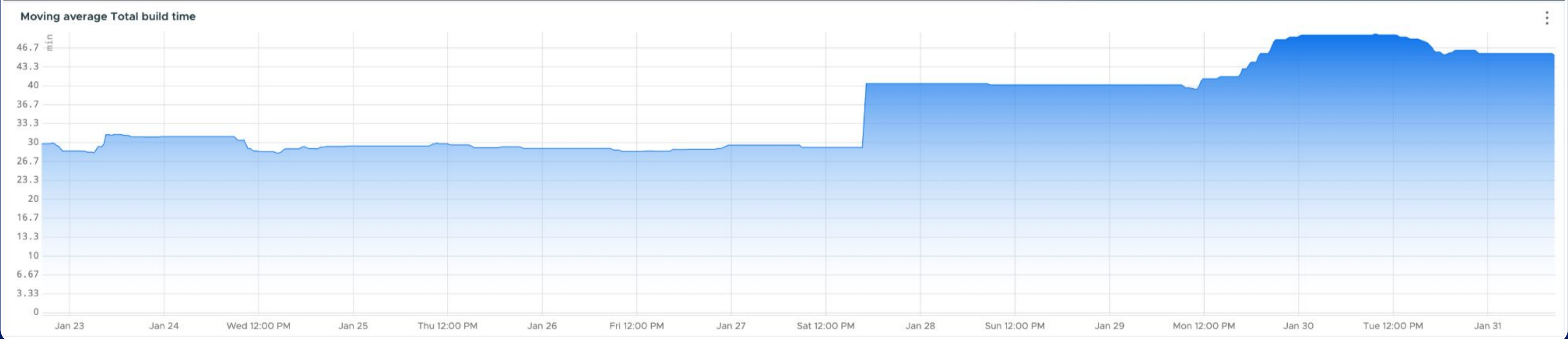
Only after that, it gets bundle in your image target and you have a standalone image.

How did this work in your past experience?



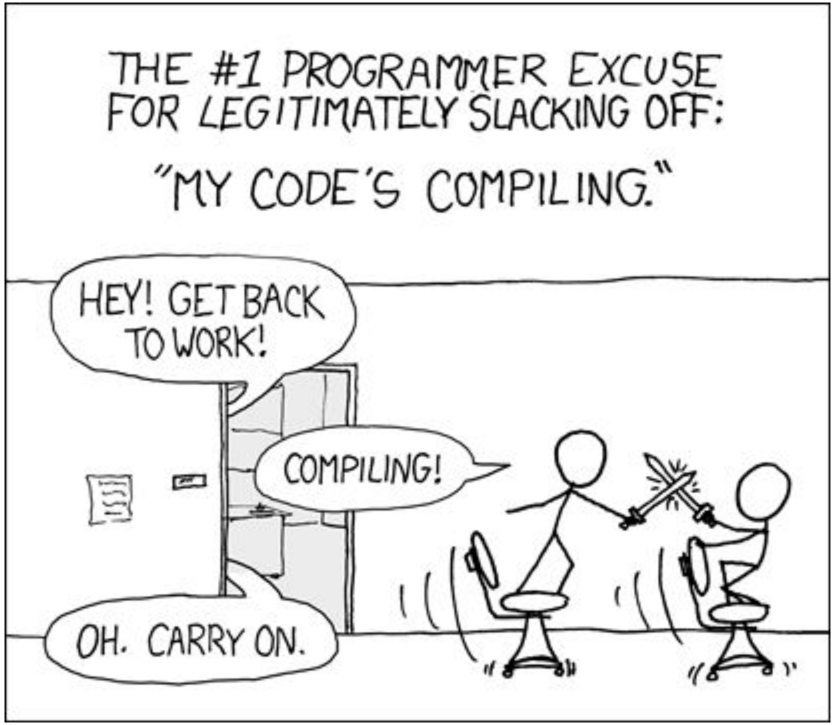
Bertrand Le Mée Apr 28th at 15:18
ah bah that's focal for you, it's blocked with cmake 3.15

and I'm betting that the newest version of grpc requires cmake 3.16.X



Dette technique

algolia



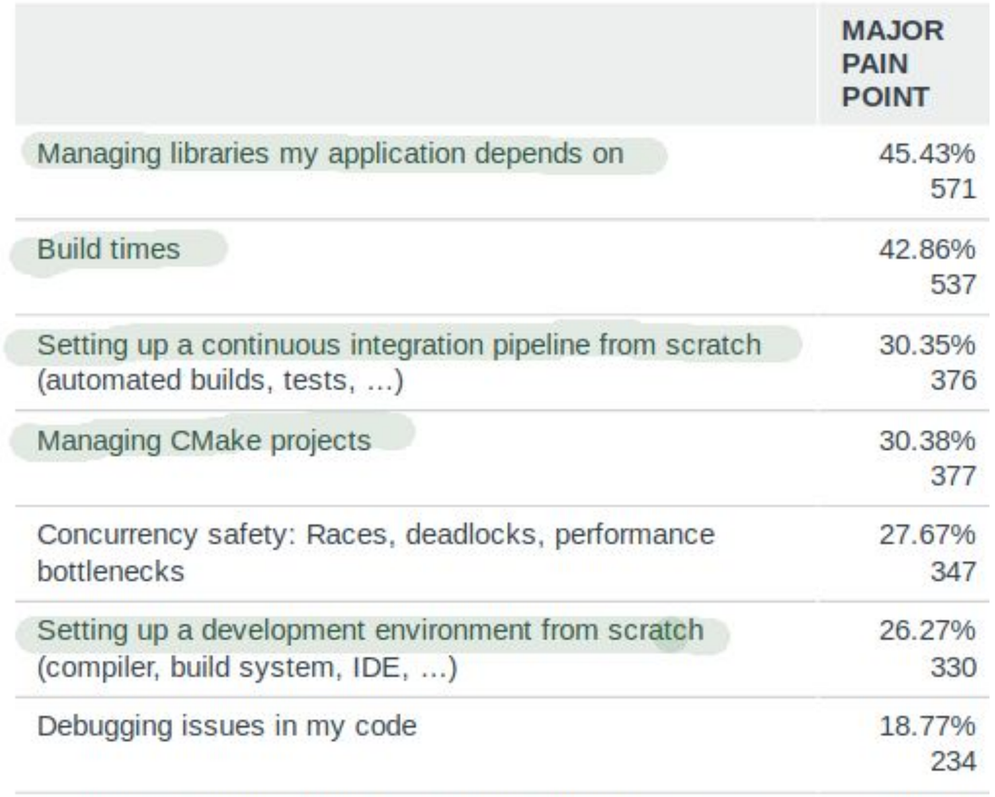
2024 Annual C++ Developer Survey "Lite"

	MAJOR PAIN POINT	MINOR PAIN POINT	NOT A SIGNIFICANT ISSUE FOR ME	TOTAL	WEIGHTED AVERAGE
Managing libraries my application depends on	45.43% 571	36.44% 458	18.14% 228	1,257	2.27
Build times	42.86% 537	37.35% 468	19.79% 248	1,253	2.23
Setting up a continuous integration pipeline from scratch (automated builds, tests, ...)	30.35% 376	42.53% 527	27.12% 336	1,239	2.03
Managing CMake projects	30.38% 377	38.20% 474	31.43% 390	1,241	1.99
Concurrency safety: Races, deadlocks, performance bottlenecks	27.67% 347	41.87% 525	30.46% 382	1,254	1.97
Setting up a development environment from scratch (compiler, build system, IDE, ...)	26.27% 330	41.80% 525	31.93% 401	1,256	1.94
Debugging issues in my code	18.77% 234	49.24% 614	32.00% 399	1,247	1.87
Parallelism support: Using more CPU/GPU/other cores to compute an answer faster	22.94% 286	36.09% 450	40.98% 511	1,247	1.82
Memory safety: Bounds safety issues (read/write beyond the bounds of an object or array)	20.48% 257	35.86% 450	43.67% 548	1,255	1.77
Memory safety: Use-after-delete/free (dangling pointers, iterators, spans, ...)	20.03% 251	34.00% 426	45.97% 576	1,253	1.74
Managing Makefiles	19.88% 235	22.42% 265	57.70% 682	1,182	1.62
Unicode, internationalization, and localization	16.56% 205	29.32% 363	54.12% 670	1,238	1.62
Type safety: Using an object as the wrong type (unsafe downcasts, unsafe unions, ...)	12.63% 158	31.18% 390	56.20% 703	1,251	1.56
Security issues: Overlaps with "safety" but includes other issues (secret disclosure, vulnerabilities, exploits, ...)	12.25% 153	30.82% 385	56.93% 711	1,249	1.55
Memory safety: Forgot to delete/free (memory leaks)	12.22% 153	27.32% 342	60.46% 757	1,252	1.52
Managing MSBuild projects	16.20% 193	18.05% 215	65.74% 783	1,191	1.50
Moving existing code to the latest language standard	9.08% 114	27.15% 341	63.77% 801	1,256	1.45

https://isocpp.org/files/papers/CppDevSurvey-2024-summary.pdf

L'outillage – un problème majeur

algolia



	MAJOR PAIN POINT
Managing libraries my application depends on	45.43% 571
Build times	42.86% 537
Setting up a continuous integration pipeline from scratch (automated builds, tests, ...)	30.35% 376
Managing CMake projects	30.38% 377
Concurrency safety: Races, deadlocks, performance bottlenecks	27.67% 347
Setting up a development environment from scratch (compiler, build system, IDE, ...)	26.27% 330
Debugging issues in my code	18.77% 234

https://isocpp.org/files/papers/CppDevSurvey-2024-summary.pdf

algolia

Durée de Compilation

Réduire le temps d'attente de la CI

Beaucoup d’options

Refactors

- Eviter les inclusions - forward declare
- Réduire les instantiations de templates
- Shared libraries + visibilité
- Pimpl idiom
- Réduire l’inter-dépendance

Changement de système

- Headers pré-compilés (PCH)
- Unity builds (jumbo builds)
- Vertical scaling
- Caches de compilation/link

# Build with clang -ftime-trace

```
>$ ClangBuildAnalyzer --all /tmp/performance performance-trace

>$ ClangBuildAnalyzer --analyze /tmp/performance/performance-trace


Analyzing build trace from 'performance-trace'...

**** Files that took longest to parse (compiler frontend):

**** Files that took longest to codegen (compiler backend):

**** Templates that took longest to instantiate:

106583 ms: std::visit<(lambda at ... (109 times, avg 977 ms)

...

**** Template sets that took longest to instantiate:

390898 ms: std::vector<$>::__swap_out_circular_buffer (12492 times, avg 31 ms)

261722 ms: std::__variant_detail::__visitation::... (2638 times, avg 99 ms)

...

**** Functions that took longest to compile:

**** Function sets that took longest to compile / optimize:

... (Some headers included everywhere e.g. vector)

430790 ms: redacted/header.h (included 180 times, avg 2393 ms), included via:...
```

Leçons apprises

- **std::visit** - à éviter dans les headers publics

Leçons apprises

- **std::visit** - à éviter dans les headers publics
- **Classes d’orchestration** - forward-declare ou pimpl

Leçons apprises

- **std::visit** - à éviter dans les headers publics
- **Classes d'orchestration** - forward-declare ou pimpl
- **Gains mineurs** - ~2mn sur 30 😢

💸 Vertical scaling

algolia

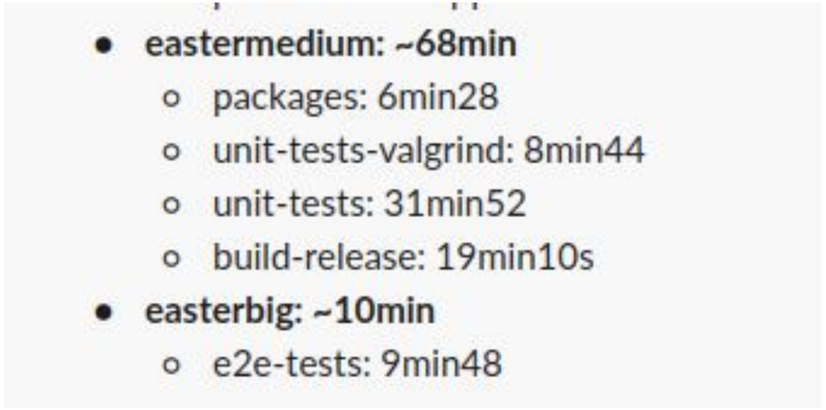
😎 **28 minutes -> 19 minutes**

✅ **Persistence**

- Utiliser ccache

❌ **Administrer**

- Hacks pour le cache
- Multi-plateforme
- Ops



- **eastermedium: ~68min**
  - packages: 6min28
  - unit-tests-valgrind: 8min44
  - unit-tests: 31min52
  - build-release: 19min10s
- **easterbig: ~10min**
  - e2e-tests: 9min48

⛅ Vertical scaling dans le cloud

algolia

Machines EC2 pour les runners GitHub

✅ **Flexibilité**
- Plateformes multiples
- Taille par runner

✅ **Ops réduites**

❌ **Machines éphémères**

sccache - un cache pour le cloud

algolia

- **Robuste** - maintenu par Mozilla
- **Local ou remote** - S3 bucket
- **Setup simple** - GH action
- **Stratégie de clef** - plateforme + branche + build type
- **Mesurer** - le pourcentage de hit avec --show-stats

```yaml
- name: ⚡ Restore sccache
  uses: actions/cache/restore@0057852bfaa89a56745cba8c7296529d2fc39830 # v4
  with:
    path: ${{ env.SCCACHE_DIR }}
    key: sccache-${{ matrix.arch.docker }}-${{ matrix.os.name }}-${{ github.ref_name }}-shared
    # Fallback restore keys: partial cache hits for first runs
    restore-keys: |
      sccache-${{ matrix.arch.docker }}-${{ matrix.os.name }}-${{ github.event.repository.default_branch }}-shared
```

RunsOn + sccache - résultats

🥳 **-80% sur le runner de build**

P50(1 mois) = ~~30+ min~~ → 6mn45

Bonus

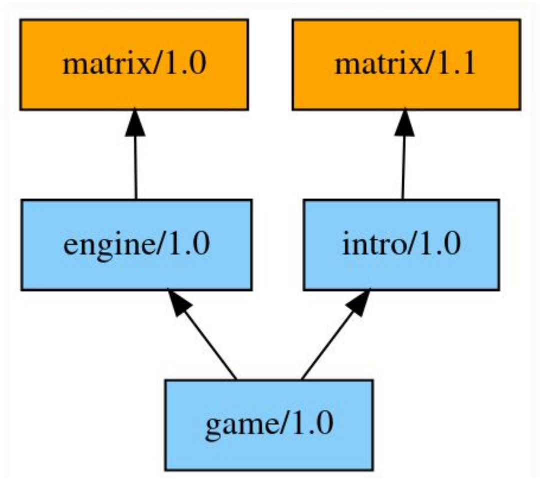
- Stabilité, multiples plateformes, ccache partagé CI/dev
- Configurabilité des runners

Conan

Ou comment déguiser un gestionnaire de dépendances en cache de compilation

Les difficultés du C++

Le"diamond problem" - avec des segfaults en bonus



La gestion des toolchains

Un paquet est spécifique à une architecture + toolchain

Choisir un gestionnaire de dépendances

Support de tous les environnements

❌ apt, brew, nix, nuget …

Choisir un gestionnaire de dépendances

Support de tous les environnements

❌ apt, brew, nix, nuget ...

Pas de réécriture majeure

❌ bazel, build2, meson

Choisir un gestionnaire de dépendances

Support de tous les environnements

❌ apt, brew, nix, nuget …

Pas de réécriture majeure

❌ bazel, build2, meson

Non-intrusif

❌ hunter, cmake, git

Choisir un gestionnaire de dépendances

Support de tous les environnements
❌ apt, brew, nix, nuget ...

Pas de réécriture majeure
❌ bazel, build2, meson

Non-intrusif
❌ hunter, cmake, git

Mainstream et open-source
✅ conan, vcpkg

Conan ou vcpkg ?

algolia

CONAN
C/C++ Package Manager

vcpkg

CONAN	vcpkg
20% du marché, en croissance	20% du marché, en croissance

Conan ou vcpkg ?

algolia

CONAN C/C++ Package Manager	vcpkg
20% du marché, en croissance	20% du marché, en croissance
1800 paquets	2600 paquets

Conan ou vcpkg ?

algolia

CONAN C/C++ Package Manager	vcpkg
20% du marché, en croissance	20% du marché, en croissance
1800 paquets	2600 paquets
JFrog	Microsoft
Artifactory, GitLab	Blob storage, GitHub

Conan ou vcpkg ?

algolia

CONAN C/C++ Package Manager	vcpkg
20% du marché, en croissance	20% du marché, en croissance
1800 paquets	2600 paquets
JFrog	Microsoft
Artifactory, GitLab	Blob storage, GitHub
Python	CMake

Conan ou vcpkg ?

CONAN C/C++ Package Manager	vcpkg
20% du marché, en croissance	20% du marché, en croissance
1800 paquets	2600 paquets
JFrog	Microsoft
Artifactory, GitLab, …	Blob storage, GitHub
Python	CMake
Paquets indépendants	**“Live at head”**

Un peu de code ?

Et dans le monde réel ?

🧹 Nettoyer → 🧑‍🍳 Cuisiner → 🛠️ Outiller → 🐸 Conan-iser → ✨ Finaliser

🧹 **CMake** - wrapper ses vendors via external_project_add

Et dans le monde réel ?

algolia

🧹 Nettoyer | 🧑‍🍳 Cuisiner | 🛠️ Outiller | 🐸 Conan-iser | ✨ Finaliser

🧹 **CMake** - wrapper ses vendors via external_project_add

🧑‍🍳 **Première recette**

- Le projet tel quel, avec un profil unique
- Retirer les flags hard-codés de CMake

Et dans le monde réel ?

algolia

🧹 Nettoyer | 🧑‍🍳 Cuisiner | 🛠️ Outiller | 🐸 Conan-iser | ✨ Finaliser

🧹 **CMake** - wrapper ses vendors via external_project_add

🧑‍🍳 **Première recette**

- Le projet tel quel, avec un profil unique
- Retirer les flags hard-codés de CMake

🛠️ **Setup la remote puis la CI** - pouvoir push/pull des artefacts

Et dans le monde réel ?

algolia

🧹 Nettoyer → 🧑‍🍳 Cuisiner → 🛠️ Outiller → 🐸 Conan-iser → ✨ Finaliser

🧹 **CMake** - wrapper ses vendors via external_project_add

🧑‍🍳 **Première recette**

- Le projet tel quel, avec un profil unique
- Retirer les flags hard-codés de CMake

🛠️ **Setup la remote puis la CI** - pouvoir push/pull des artefacts

🐸 **Tirer les dépendances**

- Bibliothèques système & open-source via conan-center
- Ecrire ses propres recettes - faire un script de bootstrap

Et dans le monde réel ?

algolia

🧹 Nettoyer → 🧑‍🍳 Cuisiner → 🛠️ Outiller → 🐸 Conan-iser → ✨ Finaliser

🧹 **CMake** - wrapper ses vendors via external_project_add

🧑‍🍳 **Première recette**

- Le projet tel quel, avec un profil unique
- Retirer les flags hard-codés de CMake

🛠️ **Setup la remote puis la CI** - pouvoir push/pull des artefacts

🐸 **Tirer les dépendances**

- Bibliothèques système & open-source via conan-center
- Ecrire ses propres recettes - faire un script de bootstrap

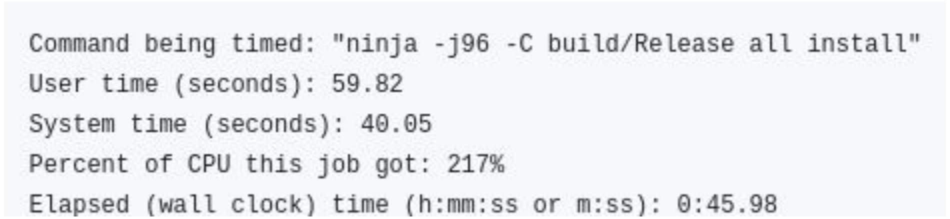
✨ **Multi-plateforme** - autres profils, autres pipelines de CI ...

Sccache + conan

🔥**Build time**

P50(1 mois) = ~~30+ min~~ → ~~6mn45~~ → **5mn40**

CI -80%, **local dev** -35%



```
Command being timed: "ninja -j96 -C build/Release all install"
User time (seconds): 59.82
System time (seconds): 40.05
Percent of CPU this job got: 217%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:45.98
```

- Des solutions simples
- Payer sa dette ouvre des opportunités
  Deps à jour, Arm64, automatisation, partage étendu...

algolia

Merci

Q & A