Mining Our Way Back to Incremental Builds for DevOps Pipelines

Shane McIntosh
Software REBELs
University of Waterloo, Canada
shane.mcintosh@uwaterloo.ca

ABSTRACT

The incremental build is a key feature of build automation tools. It still plays a key role in the build systems that underpin DevOps pipelines. Yet it is quite common for these "upper layer" automation technologies to start from a clean copy of the codebase, rendering the incremental builds inert. In this tutorial, we discuss why it is desirable to restore the incremental build features of the past. We also describe past and ongoing work that strives to make DevOps pipelines operate incrementally again. Finally, we discuss perceived barriers to adoption that our past solutions have faced.

KEYWORDS

Mining Software Repositories, DevOps Pipelines, Incremental Builds

ACM Reference Format:
Shane McIntosh. 2024. Mining Our Way Back to Incremental Builds for DevOps Pipelines. In *21st International Conference on Mining Software Repositories (MSR '24), April 15–16, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3643991.3649106

1 THE DEATH OF THE INCREMENTAL BUILD

Build systems transform sources into deliverables by orchestrating commands (e.g., compilers, test harnesses). *Build specifications* (e.g., `Makefiles`, `CMakeLists.txt`) declare platform- and configuration-sensitive dependencies, which specify whether and the order in which commands are invoked by *build tools* (e.g., `make`, `cmake`).

Since their inception [10], a key feature of software build systems has been the *incremental build.* Incremental builds traverse the graph of specified and/or implied dependencies, and invoke the subset of commands that have been directly or transitively impacted by a change set. By relying on incremental (rather than full) builds, stakeholders save time and computational resources that would otherwise be wasted performing redundant and unnecessary work.

In the past, scheduled builds that were run nightly to integrate daily activies, and personal builds that were run within IDEs or on the command line, were the primary scenarios in which build systems were used. Nowadays, builds are often performed during *Continuous Integration (CI)* [6]. CI workflows (which in turn enable DevOps pipelines) typically invoke build tools to check whether a subject change set introduces integration issues when build-triggering events occur (e.g., a pull request is created/updated).

MSR '24, April 15–16, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0587-8/24/04
https://doi.org/10.1145/3643991.3649106

Cloud-based CI services tacitly encourage full builds. Since the cloud-based machine instances (e.g., containers) that are used to perform builds are often ephemeral, it is not uncommon for CI workflows to begin by checking out a fresh copy of the code base, rendering the incremental features of build systems effectively inert. Indeed, since intermediate and output files produced by previous jobs are being deleted, the build tool is forced to consider each build within the CI context as a full (i.e., non-incremental) build.

2 THE BIRTH OF CI ACCELERATION

The literature describes several approaches that improve CI performance, which we group according to those that do not consider dependencies when making decisions and those that do.

Dependency-free solutions often make selection and/or prioritization decisions to optimize a tradeoff between the cost of CI execution and the (faster) discovery of failures. For example, Jin and Servant [14] benchmarked selection and prioritization techniques that skip or reorder entire CI jobs (e.g., CI skip [1, 2, 13, 16]) or tasks within jobs (e.g., test selection [12] and prioritization [7]). Batched execution of CI jobs has also been explored to reduce the cost of CI execution [9].

Dependency-aware solutions strive to restore incremental build behaviour by caching the results of prior builds and/or inferring dependencies. For example, organizations that can leverage a fleet of developer machines may use a build tool that leverages a shared cache of artifacts [8]. However, adoption of such technologies is not always successful [3]. Gallaba *et al.* [11] propose Kotinos—a CI service that caches the state of build containers after prior builds, and restores them for future builds to effectively build incrementally again. Approaches [4, 11] also infer dependencies by listening to file system operations that occur during full (a.k.a., "cold") builds, and leverage them to skip unaffected or irrelevant CI commands in subsequent (a.k.a., "warm") builds.

3 PERSISTENT BARRIERS TO ADOPTION

Dependency-aware approaches have the advantage of (often) behaving deterministically; however, they are not without limitations. For example, our recent work shows that even deterministic CI acceleration products can erroneously skip CI steps [18]. If an acceleration approach allows failures to permeate without being detected, stakeholders in settings where the cost of failures is high may not perceive the acceleration as beneficial enough to justify the risk.

In addition, incremental builds are only as reliable as the dependency graphs that they resolve. An *underspecified build system*, i.e., one that omits dependency expressions, may not respect an important dependency when performing tasks (in parallel), and in turn, can fail sporadically. We will discuss proposed solutions [5, 15, 17] to help identify and fix such missing dependencies.

REFERENCES

[1] Rabe Abdalkareem, Suhaib Mujahid, and Emad Shihab. 2021. A Machine Learning Approach to Improve the Detection of CI Skip Commits. *Transactions On Software Engineering (TSE)* 47, 12 (2021), 2740–2754.

[2] Rabe Abdalkareem, Suhaib Mujahid, Emad Shihab, and Juergen Rilling. 2019. Which Commits Can Be CI Skipped? *Transactions On Software Engineering (TSE)* 47, 3 (2019), 448–463.

[3] Mahmoud Alfadel and Shane McIntosh. 2024. The Classics Never Go Out of Style: An Empirical Study of Downgrades from the Bazel Build Technology. In *Proc. of the 46th Int'l Conf. on Software Engineering (ICSE)*. To appear.

[4] Talank Baral, Shanto Rahman, Bala Naren Chanumolu, Başak Balcı, Tuna Tuncer, August Shi, and Wing Lam. 2023. Optimizing Continuous Development By Detecting and Preventing Unnecessary Content Generation. In *Proc. of the 38th Int'l Conf. on Automated Software Engineering (ASE)*. 901–913.

[5] Cor-Paul Bezemer, Shane McIntosh, Bram Adams, Daniel M. German, and Ahmed E. Hassan. 2017. An Empirical Study of Unspecified Dependencies in Make-Based Build Systems. *Empirical Software Engineering* 22, 6 (2017), 3117–3148.

[6] Paul M. Duvall, Steve Matyas, and Andrew Glover. 2007. *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley Professional.

[7] Sebastian Elbaum, Gregg Rothermel, and John Penix. 2014. Techniques for Improving Regression Testing in Continuous Integration Development Environments. In *Proc. of the 22nd Int'l Symposium on the Foundations of Software Engineering (FSE)*. 235–245.

[8] Hamed Esfahani, Jonas Fietz, Qi Ke, Alexei Kolomiets, Erica Lan, Erik Mavrinac, Wolfram Schulte, Newton Sanches, and Srikanth Kandula. 2016. CloudBuild: Microsoft's distributed and caching build service. In *Proc. of the Int'l Conf. on Software Engineering (ICSE) Companion*. 11–20.

[9] Emad Fallahzadeh, Amir Hossein Bavand, and Peter C. Rigby. 2023. Accelerating Continuous Integration with Parallel Batch Testing. In *Proc. of the 31st Int'l Sym. on the Foundations of Software Engineering (FSE)*. 55–67.

[10] Stuart I. Feldman. 1979. Make—A program for maintaining computer programs. *Software: Practice and Experience* 9, 4 (1979), 255–265.

[11] Keheliya Gallaba, John Ewart, Yves Junqueira, and Shane McIntosh. 2022. Accelerating Continuous Integration by Caching Environments and Inferring Dependencies. *IEEE Transactions on Software Engineering* 48, 6 (2022), 2040–2052.

[12] Kim Herzig, Michaela Greiler, Jacek Czerwonka, and Brendan Murphy. 2015. The Art of Testing Less without Sacrificing Quality. In *Proc. of the 37th Int'l Conf. on Software Engineering (ICSE)*. 483–493.

[13] Xianhao Jin and Francisco Servant. 2020. A Cost-Efficient Approach to Building in Continuous Integration. In *Proc. of the 42nd Int'l Conf. on Software Engineering (ICSE)*. 13–25.

[14] Xianhao Jin and Francisco Servant. 2021. What helped, and what did not? An Evaluation of the Strategies to Improve Continuous Integration. In *Proc. of the 43rd Int'l Conf. on Software Engineering (ICSE)*. 213–225.

[15] Thodoris Sotiropoulis, Stefanos Chaliasos, Dimitris Mitropoulis, and Diomidis Spinellis. 2020. A Model for Detecting Faults in Build Specifications. *Proc. of the ACM on Programming Languages* 4, OOPSLA (2020), 144:1–144:30.

[16] Gengyi Sun, Sarra Habchi, and Shane McIntosh. 2024. RavenBuild: Context, Relevance, and Dependency Aware Build Outcome Prediction. In *Proc. of the 32nd Int'l Sym. on the Foundations of Software Engineering (FSE)*. To appear.

[17] Rongxin Wu, Minglei Chen, Chengpeng Wang, Gang Fan, Jiguang Qiu, and Charles Zhang. 2022. Accelerating Build Dependency Error Detection via Virtual Build. In *Proc. of the 37th Int'l Conf. on Automated Software Engineering (ASE)*. 5:1–5:12.

[18] Zhili Zeng, Tao Xiao, Maxime Lamothe, Hideaki Hata, and Shane McIntosh. 2024. A Mutation-Guided Assessment of Acceleration Appproaches for Continuous Integration: An Empirical Study of Yourbase. In *Proc. of the 21st Int'l Conf. on Mining Software Repositories (MSR)*. To appear. https://doi.org/10.1145/3643991.3644914