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

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 codebase, 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]. Gal-laba 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


