再現性のあるビルド

Mutsuha Asada
@mutsuha_asada

Kernel/VM 探検隊@つくば No3 - 2026/03/20

自己紹介

- 浅田睦葉（Mutsuha Asada）
- 筑波大学情報学群情報科学類 4 年（coins22）
- 🐣 興味
  - ビルドシステム！
    - ❄️ Nix
  - 自動ソフトウェア工学（ASE）
  - コンパイラツールチェイン、LLVM
- X: @mutsuha_asada
- のむヨーグルトブームが来ています、美味しいのむヨーグルトを教えてください！！！

再現性のあるビルド

- 近年、再現性のあるビルド（Reproducible Builds）が注目されている
  - 同一の入力に対して（ビット単位で一致した）同一の成果物が得られるビルドのこと
- たとえば、Nix や Bazel が再現性のあるビルドを実現するビルドシステム、かのように思われている
- この発表の目標
  - 実際に「再現可能なビルド」が指す概念とは何かについて知ってもらう

目次

1. モチベーション

なぜ再現性のあるビルドが必要か

- 現代の計算機環境では、同一のプログラムに同一の入力を与えて実行しても結果が変わることがある
  - 非決定的要因が原因
    - システムクロックの時刻
    - プロセス ID やスレッドの実行順序
    - 乱数、ハードウェア依存命令
- どんな第三者が再ビルドしても同一の結果を得たい

なぜ結果を一致させたいのか

- 配布されたバイナリが本当に公開されたソースから作られたのか、第三者が検証したい
  - サプライチェーン攻撃への耐性
  - 配布物への信頼の向上
- たとえば、開発者が悪性のツールチェインに依存したり、公式のリリースビルド CI が侵害されたりするなどして、余計なオブジェクトファイルやパッチ、マルウェアが差し込まれる場合に、公式が配布するバイナリが汚染されていることを第三者が検証できる

セキュリティ以外のメリット

- 再現性のあるビルドを導入することは、サプライチェーン攻撃などのセキュリティ対策以外のメリットもある
- 再現性を取るための過程で、レアなビルドエラーや環境依存などが炙り出され、以下のメリットが得られる
  - QA、デバッグの効率が上昇
  - 成果物の差分が小さくなる
  - キャッシュ効率が改善する

2. 再現性

再現性とは

- 同じソースコード、ビルドマニフェスト、ビルド環境から、指定された成果物を第三者が再生成できること
- ただし
  - 未来永劫ビルドできること、ではない
  - Nix や Bazel を使えば自動的に達成される訳でもない
- 再現性を保証することは、（Reproducible Containers - Omar S. Navarro Leija et al. によれば）決定性（determinism）と移植性（porability）を満たすこと
  - それぞれの性質を見ていく

データフロー決定性

- 同一のマシン上では、すべての read 操作が毎回同じ値を返すという性質を「データフロー決定性」と呼ぶ
- データフロー決定性が成り立つと、以下のような性質が得られる
  - 全てのプロセスが終了した後のファイルシステムの状態が常に同一
  - 標準出力、標準エラー出力に出力されるメッセージも常に同一
- しかし厳密には決定化できない外部要因が存在する

準決定性

- たとえばディスク容量が枯渇した場合、クラッシュしてしまい決定的ではなくなってしまう…
- そこで、準決定性（quasi-determinism）という性質を導入する
  - 同一の実行を 2 回行ったとき、データフロー決定性であるか、クラッシュすることが保証されるとき、そのビルドは準決定的であるという
- 後述するが、完全な再現性を保証することは非常にコストが高く、どれくらいコストをかけて再現性を保証するか、という話になってくる
  - 準決定性は決定性をできるだけ保証したい場合の現実的な落とし所

移植性

- データフロー決定性が異なるマシン間でも成立することを「移植性」と呼ぶ
  - CPU のマイクロアーキテクチャや OS のバージョンが異なっていても同一の結果が得られることが求められる
  - たとえば、仮想的に CPU 情報や OS 情報を提示したり、CPU 命令をエミュレートすることで達成できる
- データフロー決定性と、移植性を両方満たしたときに、再現性を満たすと言える

既存の手法は決定性を満たすか？

- コンテナ技術（例: Docker）
  - ホスト OS の詳細（ファイルシステムの mtime、ctime、カーネル構成）や、プロセッサのマイクロアーキテクチャの詳細（CPU 命令レベル）などの情報が内部から直接見える
  - → 決定性も移植性も持たない
- 仮想マシン（以下の 3 つの理由により決定性を持たない）
  - VM 上のスレッドスケジューラや I/O 割込みがホスト OS のタイミング依存
  - VM の RTC（Real Time Clock）や TSC（Time Stamp Counter）はホストの時計や NTP などに依存
  - VM ごとにエミュレートされるハードウェア構成が異なる

グラデーション

- 現実的には再現性を目指す仕組みには、何をどこまで固定し、誰がどこまで検証できるかの段階がある
- 一般的なビルド
  - ソースコードは固定されていても、環境は暗黙
- ロックファイル、コンテナ、仮想マシン
  - 依存関係や OS イメージの大部分を固定できる
- サンドボックス、Hermetic build（Nix や Bazel など）
  - 入力を明示して暗黙の入力を防ぐことができる
- Reproducible Build
  - 同じソース、ビルドマニフェスト、ビルド環境から第三者が同じ成果物を再生成できる

3. まとめ

まとめ

- 再現性のあるビルドとは、同じソースコード・ビルドマニフェスト・ビルド環境から第三者が同じ成果物を再生成できることである
- その価値は、配布物が本当に公開されたソースから作られたかを検証できることにある
- ただし、未来永劫ビルドできることを保証するものでも、Nix や Bazel を使えば自動的に達成されるものでもない
- 現実には再現性はグラデーションであり、暗黙の入力を減らし、入力を明示・固定・検証するほど再現性に近づく

興味がある方へ

- Reproducible Containers（Omar S. Navarro Leija, et al.）を読んでください
  - ASPLOS'20 に採択されています
  - https://github.com/dettrace/dettrace
- 時間がない場合は↑を元にして書いたゼミ資料を読んでください。どのように再現性を取るかについても触れています
  - https://speakerdeck.com/momeemt/reproducible-containers-asplos20?slide=4