semaphore

# CI/CD for Monorepos

Sample

Effectively building, testing, and deploying code with monorepos.

# CI/CD for Monorepos

Effectively building, testing, and deploying code with monorepos

Semaphore

# Contents

This book is open source: https://github.com/semaphoreci/book-monorepo-cicd

Published on the Semaphore website: https://semaphoreci.com

Sep 2022: First edition v1.0 (revision c5d9951)

Share this book:

> *I've just started reading "CI/CD for Monorepos", a free ebook by @semaphoreci: https://bit.ly/3yopUT2 (Tweet this!)*

## Preface

A monorepo is a new name for an old idea — placing a bunch of software projects into the same code repository. Organizations that overcome the technical challenges associated with adopting monorepos enjoy significant benefits:

- **Cultural** — increased bandwidth of knowledge transfer and a higher level of collaboration among teams.
- **Technical** — common coding and tooling standards, simplified dependency management, and configuration reuse.

Big companies like Google, Facebook, Twitter, and Airbnb have been using monorepos for years. Today, there are a growing number of smaller teams adopting monorepos.

Why the change now? On the frontend side, the proliferation of JavaScript-based tools is such that it is possible to develop very complex applications in a single programming language. Architects of frontend projects now face the problems of separating concerns and avoiding code duplication — and they have a good set of tools to solve these problems by working in a monorepo.

On the backend side, serverless and microservices-based architectures drive developers to logically isolate their code into small units. Most of these services are written with the same set of tools and coding standards, and built, configured, and deployed in the same way. Placing them into a monorepo is an efficient way of avoiding duplicate configurations and processes.

At Semaphore, we have observed this trend in a growing number of teams and have solved the technical challenge of running effective CI/CD pipelines for monorepos.

Using traditional CI/CD tools in the monorepo context, developers essentially need to build, test, and deploy all services all the time. Using Semaphore, developers run dynamic CI/CD workflows that run the right pipelines at the right time. This gives product teams more time to focus on building the next great feature.

## Who Is This Book for, and What Does It Cover?

This book is intended for software engineers who are either exploring using a monorepo for software development or looking to optimize the CI/CD process for their monorepo.

By showing what it takes to build a monorepo-first CI/CD pipeline that saves time and speeds up software development cycles, we hope that CTOs and other engineering leaders will be able to determine if monorepos are the way forward for their companies and teams.

Chapter 1, "Introduction to Monorepo", introduces the basics and relates stories about other companies that have successfully migrated to a monorepo. This chapter will help you decide if a monorepo is right for you.

Chapter 2, "Continuous Integration", explains what you need to know about setting up a CI pipeline that builds and tests only the code that changes.

In chapter 3, "Continuous Integration Demo", we apply the knowledge gained so far into building and testing a demo monorepo with working microservices.

Chapter 4, "Continuous Deployment", describes how to expand the CI pipeline with continuous deployments. We'll learn how to implement a continuous deployment pipeline on top of a working project.

## How to Contact Us

We would very much love to hear your feedback after reading this book. What did you like and learn? What could be improved? Is there something we could explain further?

A benefit of publishing an ebook is that we can continuously improve it. And that's exactly what we intend to do based on your feedback.

You can send us feedback by sending an email to learn@semaphoreci.com.

Find us on Twitter: https://twitter.com/semaphoreci

Find us on Facebook: https://facebook.com/SemaphoreCI

Find us on LinkedIn: https://www.linkedin.com/company/rendered-text

## About the Author

**Pablo Tomas Fernandez Zavalia** is an electronic engineer and writer. He started out developing for the City Hall of Buenos Aires (buenosaires.gob.ar). After graduating, he joined British Telecom as head of the Web Services department in Argentina. He then worked for IBM as a database administrator, where he also did tutoring, DevOps, and cloud migrations. In his free time, he enjoys writing, sailing, and board games. Follow Tomas on Twitter at @tomfernblog.

## About the Editor

**Marko Anastasov** is a software engineer, author, and entrepreneur. Marko co-founded Rendered Text, the software company behind the Semaphore CI/CD service. He worked on building and scaling Semaphore from an idea to a cloud-based platform used by some of the world's best engineering teams. Follow Marko on Twitter at @markoa.

# 1 Introduction to Monorepos

Monorepos can be a great force for fostering rapid development workflows. But, are they the right fit for you, your team, and your company?

## 1.1 What Is a Monorepo?

Not everyone agrees on a single definition for *monorepo*. Some may only accept the term when it applies to companies hosting *all* their code in a single repository. Google is the most famous example of this; their monorepo is theorized to be the largest code repository in the world, which has thousands of commits per day and exceeds 80 TBs in size.

More relaxed definitions will say that *a monorepo is a version-controlled code repository holding a number of independently-deployable projects*. While these projects may be related, they are often separate, logically-independent, and run by different teams. For instance, Airbnb has two monorepos: one for the frontend code and one for the backend code. In this way, a company or organization can utilize multiple monorepos.

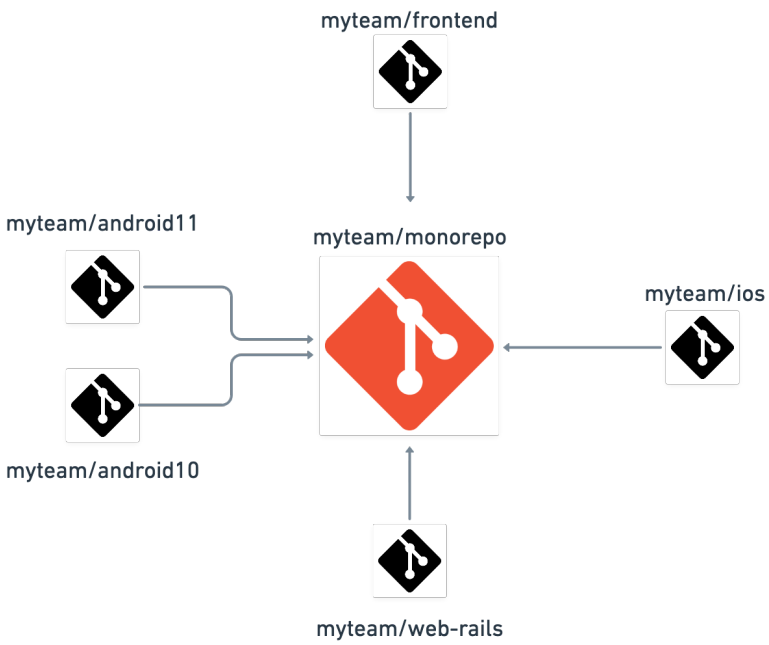
Monorepos are sometimes also called *monolithic repositories*, but they should not be confused with *monolithic architecture*, a software development practice for writing centralized applications using a single codebase. To give one example to these kinds of architectures, think of a Ruby on Rails application handling websites, API endpoints, and background jobs all at once.

## 1.2 Monorepos vs. Multirepos

The opposite of the monorepo is a *multirepo*, or simply *repos*, where each project is held on a completely separate, version-controlled software repository. Multirepos come naturally — it's what we do when starting a new project. After all, who doesn't like starting fresh?

Moving from multi to monorepo is merely a matter of moving all your projects into a single repository.

```
$ mkdir monorepo
$ git init
$ mv ~/src/app-android10 ~/src/app-android11 ~/src/app-ios .
$ git add -A
$ git commit -m "My first monorepo"
```




Of course, this is just to get started. The hard work comes later, when we get into refactoring and consolidation. To enjoy the full benefits of a monorepo, all shareable code should be moved outside of each project folder and into a common location.

Multirepos are not a synonym for *microservices*. In fact, having one does not require using the other. Later, we'll discuss companies using monorepos *with* microservices. A monorepo can host any number of microservices as long as you carefully set up your Continuous Integration and Delivery (CI/CD) pipeline[1] for deployment.

## 1.3 What Monorepos Bring to the Table

At first glance, the choice between monorepos and multirepos might not seem like a big deal. On closer inspection, however, it's a decision that deeply influences how you and your team interact.

Monorepos have the following benefits:

- **Visibility**: everyone can see everyone else's code, leading to better collaboration and cross-team contributions. Any developer can fix a bug

[1]CI/CD Pipeline, A Gentle Introduction - *https://semaphoreci.com/blog/cicd-pipeline*

in your code before you even notice it.

- **Simpler dependency management**: sharing dependencies is trivial. There's little need for a complex package manager setups as all modules are hosted in the same repository.
- **Single source of truth**: one version of every dependency means there are no versioning conflicts and no dependency hell.
- **Consistency**: enforcing code quality standards and a unified style is straightforward when you have your entire codebase in one place.
- **Shared timeline**: breaking changes in APIs or shared libraries are immediately exposed, forcing different teams to communicate and join forces. Monorepos keep everyone invested in keeping up with changes.
- **Atomic commits**: atomic commits make large-scale refactoring possible. In theory, a developer can update several packages or projects at once in a single commit. In practice, these types of changes are usually rolled out in stages, not all at once.
- **Implicit CI**: continuous integration is guaranteed as all the code is already integrated into one place.
- **Unified CI/CD process**: you can use the same CI/CD deployment process for every project in the repo.

## 1.4 Technical Challenges

As monorepos grow, we reach design limits in version control tools, build, systems, and continuous integration solutions. These problems can make a company go the multirepo route:

- **Bad performance**: monorepos can be difficult to scale up. Commands like `git blame` take unreasonably long, IDEs begin to lag, and testing the whole repo for every change becomes infeasible.
- **Broken main/master**: a broken master affects everyone working in the monorepo. This can be seen as either disastrous or as a good motivation to keep tests clean and up to date.
- **Learning curve**: the learning curve for new developers is steeper if the repository spans many tightly-coupled projects. Keep in mind, however, that the same can be the case with multi-repos.
- **Large volumes of storage**: monorepos can reach unwieldy sizes and very large quantities of commits per day.
- **Ownership**: maintaining ownership of files is more challenging. Systems like Git or Mercurial don't feature built-in directory-level permissions.
- **Code reviews**: notifications can get very noisy. For instance, GitHub

sends notifications about PRs to every developer in the repository.

You may have noticed that these problems are mostly technical. Some of them can be mitigated by adopting the *trunk-based development* model, which encourages engineers to collaborate in a single branch — the trunk — and proposes limiting the lifespan of topic branches to a minimum.

## 1.5 It's Not (Only) about Technology

Choosing a repository strategy is not only a technical matter but also about how people communicate. As stated by Conway's Law, communication is essential for building great products:

> Any organization that designs a system will produce a design whose structure is a copy of the organization's communication structure.
>
> — Melvin E. Conway

While multirepos allow each team to manage their projects independently, they also put up communications barriers. In that way, they can act as blinders, making developers focus only on the part they own, forgetting the overall picture.

A monorepo, on the other hand, works as a central hub, a market square of sorts where developers, engineers, testers, and business analysts meet and talk. Monorepos encourage conversations while helping bring silos down.

## 1.6 Notable Monorepo Adopters

Open-source projects, by their nature, have more freedom to experiment and feel greater pressure to self-organize. For three decades, FreeBSD has used CVS and later subversion monorepos for development and package distribution. Other notable projects with monorepo support or that are monorepos themselves are Babel, Google's Angular, Facebook's React and Jest, and Gatsby.

Comercial companies have also posted about their journey towards monorepos. Besides the big ones like Google, Facebook, or Twitter, we find some interesting cases such as:

- Segment.com[2]: a company offering an event collection and forwarding service. Initially, they used one repository per customer. As the number of customers increased, they moved their 140 repositories into a single one.

[2]Goodbye Microservices - *https://segment.com/blog/goodbye-microservices/*

They migrated all the services and dependencies into their monorepo. While the transition was successful, it was very taxing as they had to reconcile shared libraries and test everything each time. Still, the end result was reduced complexity and increased maintainability.

- Airbnb[3]: initially ran on Ruby on Rails. Their "monorail" accompanied the company's exponential growth, until it didn't. Eventually, it was obvious that the rate of changes and number of commits was too much for a single repository. After some debate, they chose to split development into two monorepos: one for the frontend and one for the backend. Both comprise hundreds of services, the documentation, Terraform and Kubernetes resources for deployment, and all the maintenance tools.
- Pinterest[4]: has an ongoing three-year-long migration. The plan is to move more than 1300 repositories into only four monorepos and then consolidate hundreds of dependencies into a monolithic web application. The objective is to get a more uniform build process and higher quality standard. Automation, simplification, and standardization of release practices allowed them to cut down on boilerplate and let developers focus on writing code.
- Uber[5]: their build system used to be a combination of the Golang toolchain and Make. As they moved their mobile development to the monorepo and the number of files reached the 70 thousand mark, Make no longer fulfilled their needs. They elected to adopt Bazel, an offshoot of Google's build system, designed for scalability and featuring incremental builds, to which they ended contributing several patches and improvements. According to Uber, their monorepo is likely one of the largest Go repositories running on Bazel.

## 1.7 Investing in Tooling

If we have to take only one lesson from all these stories, it is that proper tooling is key for effective monorepos. Building and testing need to be rethought: instead of rebuilding the entire repo on each update, we can use smart build systems that understand the structure of the projects and work only on the parts that change.

[3]From Monorail to Monorepo, Airbnb's journey into Microservices - *https://www.youtube.com/watch?v=sakGeE4xVZs*

[4]Pinterest's journey to a Bazel monorepo - *https://www.youtube.com/watch?v=r5KHQnS6uP8*

[5]Building Uber's Go Monorepo with Bazel - *https://eng.uber.com/go-monorepo-bazel/*

On a high level, a smart build system would need to:

1. Determine which files changed due to commits since the last build.
2. Find all the projects and their dependencies affected by the changes.
3. Build these projects, ideally using some form of caching.
4. Run tests based on affected code.
5. Deploy the projects that have changed into staging or production.

Most of us, however, don't have Airbnb's resources. So, what can we do? Fortunately, many larger companies have open-sourced their build systems:

- Bazel: released by Google and based partly on their homegrown build system (Blaze). Bazel supports many languages and is capable of building and testing at scale.
- Buck: Facebook's open-source fast build system. Supports differential builds on many languages and platforms.
- Pants: The Pants build system was created in collaboration with Twitter and Foursquare. For the moment, it supports only Python, but more languages are on the way.
- RushJS: Microsoft's scalable monorepo manager for JavaScript.

Monorepos seem to be getting more attention, particularly in JavaScript, as shown by these projects:

- Lerna: monorepo manager for JavaScript. Integrates with popular frameworks like React, Angular, or Babel.
- Yarn Workspaces: installs and updates dependencies for Node.js in multiple places with a single command.
- ultra-runner: scripts for JavaScripts monorepo management. Works with Yarn, pnpm, and Lerna. Supports parallel building.
- Monorepo builder: installs and updates packages across PHP monorepos.
- NPM: since version 7, has support for workspace.

## 1.8 Scaling up Repositories

Source control is another sticking point for monorepos. These tools can help you scale up repositories:

- Virtual Filesystem for Git (VFS): adds streaming support for Git. VFS downloads objects from Git repositories as needed. This project was originally created to manage the Windows codebase (which is the largest Git repository). Works only in Windows, but MacOS support has been announced.

## Download the full ebook for free

We hope you have enjoyed this small sample of the ebook.

Download the the full ebook for free here:

https://semaphoreci.com/resources/monorepo-cicd