Vercel

Improve build speed and developer velocity with monorepos

TRUSTED BY THE BEST FRONTEND TEAMS

NETFLIX HashiCorp Meta The Washington Post Auth0 Uber Tripadvisor

//Contents

What are monorepos?

What are monorepos?

Championed by Google and Meta, monorepos are codebases that can contain multiple projects in a single code repository. This kind of architecture enables teams to use multiple programming languages and frameworks all in one space.

For frontend teams, monorepos can improve collaboration, transparency, and overall productivity through code sharing, discoverability, and standardization.

While monorepos offer several workflow benefits to developers, they present challenges, too—especially for smaller teams who can't afford to build, maintain, and/or configure sophisticated build systems large organizations have.

In this brief, we're going to explore those challenges and show how your team can overcome them with Turborepo, a high performance build system that makes monorepos accessible to everyone.

Why do teams use monorepos?

Monorepos can contribute to a better, more efficient workflow. From faster development to easier collaboration, they enable developers to ship with confidence.

Let's start with shared code and visibility. Monorepos allow a team to centralize around a single source of truth so it's easier to search, share, and reuse code across systems, applications, libraries, and utilities. As code reuse and visibility improves, so does code consistency.

Monorepos support atomic changes, or the ability to update multiple projects in a single commit. This also helps ensure consistency across products and features, and prevents the pain of trying to coordinate commits across your entire codebase.

Faster development

Easier collaboration

Smarter workflow

Co-locating your code, issues, pull requests, deployment, automation, testing, and more reduces context switching and noise. **What does this mean for your team's productivity?** Everything from onboarding new hires to shipping new features gets a whole lot faster.

If monorepos are so great, why doesn't everyone use them?

For all the benefits of monorepos, they aren't without challenges. While some larger organizations have the budget and resources to overcome these challenges, leaner teams often feel this architecture is out of reach.

With the right tools, anyone can unlock the power of monorepos.

Enter Turborepo

Turborepo takes the lessons and development workflows from the giants of the Web and brings them to teams of all sizes. As an open-source project, it lowers the barriers to entry for using monorepos and making them accessible to everyone.

MAKESWIFT

"CI/CD times went from 20 minutes to 8 minutes once we started using Turborepo."

Multiple releases launched ahead of schedule

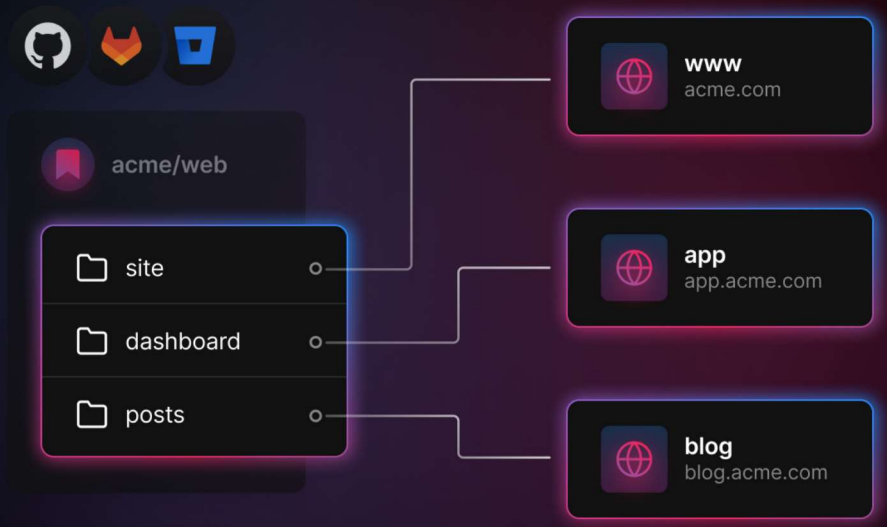
65% Faster CI/CD pipeline

2.5X Faster build speeds

Miguel Oller Founder and CTO, Makeswift

Turborepo is a high-performance build system for JavaScript and TypeScript codebases that abstracts away the complex configuration needed for monorepos, giving you a world-class development experience without the maintenance burden.

Before you can fully understand the power of tools like Turborepo, let's unpack the **top monorepo challenges that these tools solve for.**



Monorepo challenges

Monorepo challenges

Speed

1

Ask a developer their biggest headache about monorepos and they'll tell you that they need constant maintenance, break often, and often result in painfully slow builds. With so many interdependencies and packages, builds take too many steps and can bring iteration to a standstill. And with any change, the entire monorepo has to be rebuilt. (You no longer have the benefit of being able to build one small piece of the application.)

When it comes to caching builds, that's equally as painful. Constant reruns waste time and resources. Ideally, your developers would only execute work that hasn't been done before. This process of only building what's changed is called **incremental builds.**

To do that, your team needs to write automation that knows and understands your codebase at a deeper dependency and task level. What's more, they then have to maintain it. The end result: instead of focusing on your product, developers must also create and maintain an additional build system.

This is where the appeal of monorepos fades. Developers feel like they're fighting—and responsible for maintaining—a system that should be making their lives easier.

Monorepo challenges

Orchestration

2

Another significant challenge with monorepos is when it comes time to schedule tasks. Consider testing: developers report that testing is one of their main hurdles to shipping faster.

However, testing is particularly complicated with monorepos because both test and often build/compile tasks must be completed in the right order to ensure that all dependencies are built from source and available at the correct time.

Like builds, testing an entire codebase takes time. The same concept of incrementalism applies: **If you're only testing for one small change, you only want that change to be tested and re-deployed. You also want to know that change won't negatively impact other applications in your monorepo. Easily said—not so easily done.**

For example, say you change some copy and forget to update the snapshot test. The test suite fails and the entire thing has to be rerun. In the case of monorepos, it's a trivial change that ends up taking a lot of time to fix. Identifying and rerunning the right test, at the right time, and confirming it won't impact other tasks or code requires high levels of orchestration and dependency analysis.

Monorepo challenges

Complexity

3

Monorepos are complicated. There's a lot to manage and—equally—a lot that can go wrong. Mapping what works for a single package or application to multiple packages and applications is an enormously complex task that's not always obvious or well documented. It also requires custom coordination and automation scripts to get it to work correctly.

Even if you have a tool that allows you to run a script across different packages, the existing tooling often isn't flexible when a one-off package needs to do something a little differently.

It's either all or nothing. Instead of focusing on building and shipping the product, this lack of standardization leaves developers continuously configuring and piecing together solutions—which then also need to be maintained.

Your team needs a tool that is flexible to meet new needs or different configurations without forcing them to start from scratch.

Shipping takes too long

After speaking with many development teams, one thing became clear—deployments are simply slower than they should be with monorepos. Teams are installing and often building more than what's needed with each change. The benefits of monorepos are almost within reach, but developers continue to struggle to solve for challenges like speed, orchestration, and complexity.

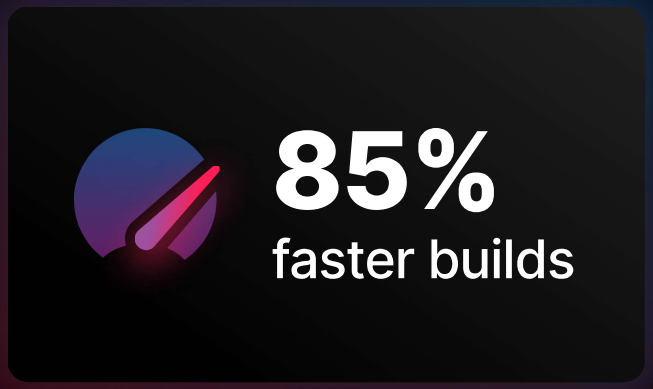
If your team has gone down this path, you've probably explored different solutions like finding a tool or building one in house. Even so, you're still seeing builds take 30 minutes. Or maybe you've investigated Bazel and Buck (complex build systems from Google and Meta respectively). But the enormous constraints on your codebase seem risky, especially when the migration would pull your team away from shipping your product.

Monorepos shouldn't be this complicated.

Instead, Turborepo makes them accessible—with a high-performance build system you can add in a day, not weeks or months.

The TURBOREPO solution

The Turborepo solution



Turborepo removes the complex maintenance burden holding developers back from adopting monorepos—and makes build times faster.

Let's explore how Turborepo helps teams with:

- Incremental builds
- Remote caching
- Parallel task running

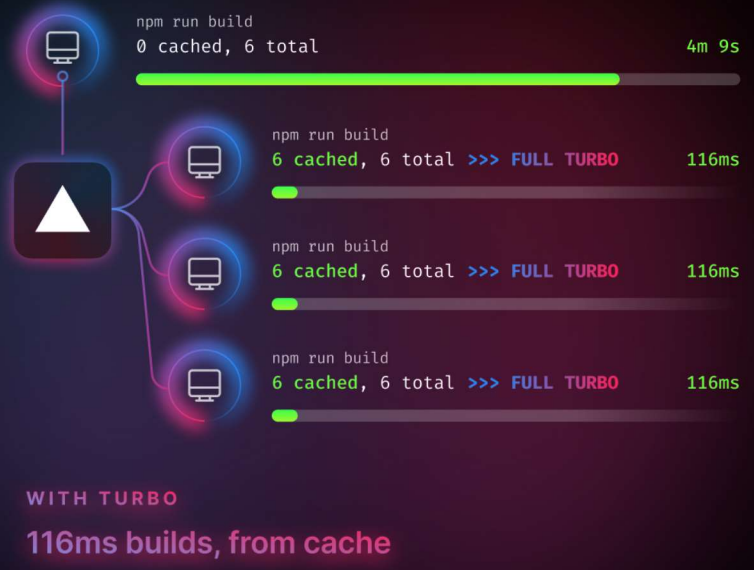
Incremental builds

Building once is painful enough. By analyzing your codebase, Turborepo can determine what's new and what's impacted by a given change to determine the minimum amount of work that needs to be done.

Remote caching

If incremental builds determine what work to do, remote caching determines if this work has been done before. Another way to think of remote caching is like a shared drive for your builds. That means you can share the Turborepo cache across your whole team and CI, and you won't need to recompile, retest, or re-execute your code if it's unchanged.

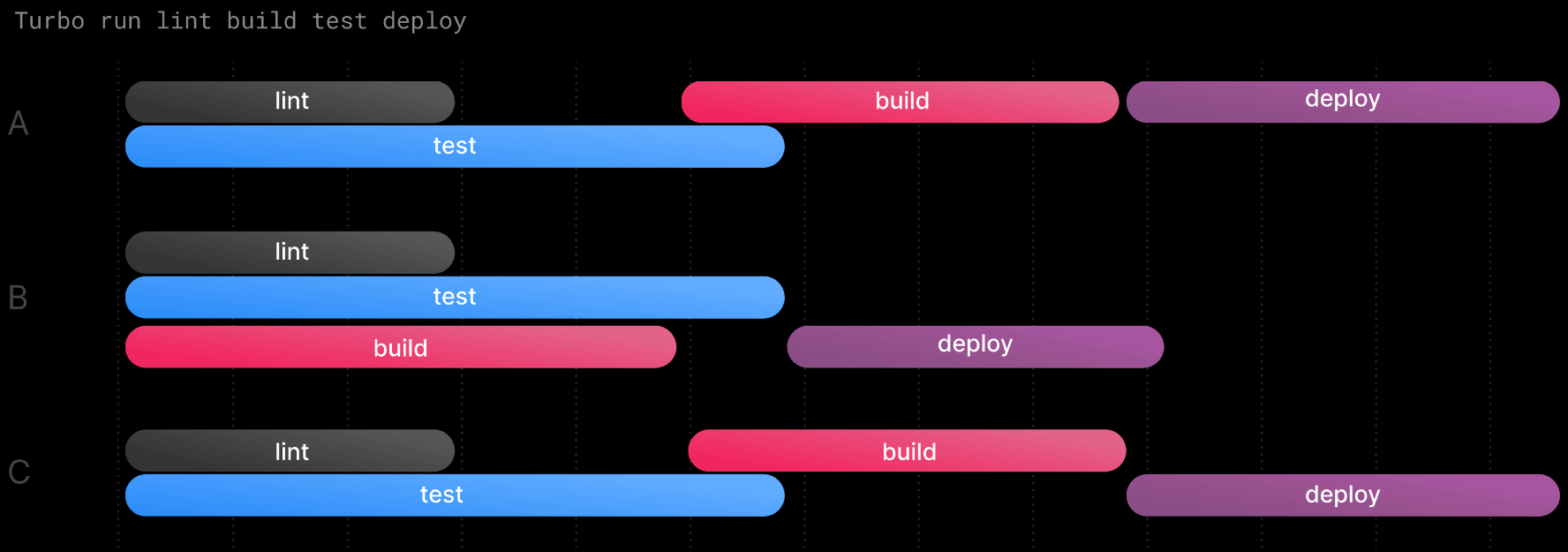
Aside from saving space on your machine, it also results in even faster, more incremental builds. Plus, if you only have one small change to one piece of code, you only need to test that code, not your whole codebase.



Parallel task running

To speed things up even further, Turborepo helps developers split builds into parallel operations, so they can run multiple tasks at the same time. This is unique to Turborepo—while other JavaScript build tools run things in dependency-first order, Turborepo can determine how different tasks relate to one another.

Parallel tasks save an enormous amount of time in the build process and will help your team iterate and ship faster, without worrying that one small change will hold up the whole process.



Watershed

"With Turborepo, we were able to give each workspace its own build, test, and typecheck scripts and not worry about manually managing when they execute—Turborepo handles the pipelining and caching. Turborepo's remote caching has drastically sped up our CI runs when a code change only touches one or a few workspaces."

Spike Brehm Software Engineer, Watershed

on Vercel

Turborepo on Vercel

All performance. No overhead.

Turborepo is open source and works well with most development tools, but its full power is realized on Vercel, the end-to-end development platform. Turborepo works out-of-the-box on Vercel and is easy to set up with zero configuration needed.

This means teams can adopt and use features like remote caching right away. Since Turborepo joined Vercel, we've seen development teams of all sizes adopt Turborepo for faster builds and save an average of 76.8 hours per month by remotely caching their deployments on Vercel.

TOTAL COMPUTE MINUTES SAVED BY TURBOREPO. AND COUNTING.

Turborepo on Vercel

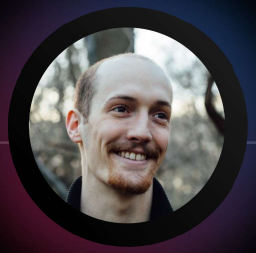
Monorepos keep your code organized. Turborepo makes it fast.

Monorepos can be extremely useful for productivity and collaboration, but the tooling can be a nightmare. It's become completely normal to spend entire days on tooling alone—tweaking configs, writing one-off scripts, and stitching things together. In the end, your development cycles take longer, which means products are released slower than they can and should be.

We need a fresh take on the whole setup: A toolchain that works for you and not against you, with sensible defaults, and even better escape hatches. One that's built with the same techniques used by large teams, but in a way that doesn't require a PhD to learn or staff to maintain.

With Turborepo on Vercel, we're doing just that: abstracting the complex configuration needed for most monorepos into a single cohesive build system—giving you a world-class development experience without the maintenance burden.

STATELY



"Turborepo has saved us 67 hours of CI since we adopted it. That's for a team of only four full-time developers."

Matt Pocock Lead Developer, Stately.ai