MONOREPOS.2022

THE GOOD, THE BAD, AND THE UGLY

What is Monorepo?

WIKIPEDIA
The Free Encyclopedia

In version control systems, a monorepo ("mono" meaning 'single' and "repo" being short for 'repository') is a software development strategy where code for many projects is stored in the same repository

What means..

All packages you may want to store in separated git repos, but fancy packed in one place (git repo)

When we may want to use it?

- We want easily code base management
- We have a huge codebase with cross depends on packages
- We have several `applications` and the `packages` we use inside every application
- We don't want to deal with `yarn link`, etc if we need to fix something inside our package
- We want flexible versioning, publishing, and changelog generation
- **Just you can** 😎

The Players

Based on State of JS 2020 most popular projects are Lerna (12,7%) and NX (15,8%). Other tools don't present

npm / Workspaces

- ✓ Link dependencies
- ✓ Run commands across the packages
- ✗ Automated packages publishing
- ✗ Automated version managing

Low-level primitive used by other tools

Lerna (★31.4k)

- ✓ Link dependencies
- ✓ Run commands across the packages
- ✓ Automated packages publishing
- ✓ Automated version managing
- ✗ Caching

TURBOREPO (★5.3k)

- ✓ Link dependencies
- ✓ Run commands across the packages
- ✗ Automated packages publishing
- ✗ Automated version managing
- ✓ Caching

Nx (★10.3k)

- ✓ More than just monorepo
- ✗ Automated packages publishing
- ✗ Automated version managing
- ✓ Caching

BIT (★14.6k)

- ✓ More than just monorepo
- ✓ Good for micro frontends

Other..

Before we go further..

- ✓ Smart rebuilds of affected projects
- ✓ Computation caching
- ✓ Code sharing and ownership management
- ✓ High-quality editor plugins

- ✓ Rich plugin ecosystem
- ✓ Workspace visualizations
- ✓ Powerful code generators
- ✓ Consistent dev experience for any framework

Give it a try

✗ Be ready for the challenges

The Winner..

TURBOREPO

Building once is painful enough, Turborepo will remember what you've built and skip the stuff that's already been computed.

Turborepo looks at the contents of your files, not timestamps to figure out what needs to be built.

Share a remote build cache with your teammates and CI/CD for even faster builds.

Execute builds using every core at maximum parallelism without wasting idle CPUs.

Define the relationships between your tasks and then let Turborepo optimize what to build and when.

Generate build profiles and import them in Chrome or Edge to understand which tasks are taking the longest.

The Digits..

	`build-packages` *First Run*	`build-packages` *Re-run*	`build-app` *First Run*	`build-app` *Re-run*
	✓ 5.29 sec	✗ 4.45 sec	88.74 sec	✗ 36.47 sec
TURBOREPO	✗ 19.25 sec	✓ 2.63 sec	90 sec	✓ 1.25 sec - 6 sec

*The repo contains **one** React application and **13** packages. One of the packages is the UI Library Kit which contains **102** React components*

The Changes ..

```
 "devDependencies": {
    - "concurrently": "^5.0.0",
    - "lerna": "^3.16.4",
    + "turbo": "^1.0.24"
  },
  "scripts": {
    - "build-packages": "lerna run build --parallel --scope package-1 --scope package-n ... 11 more times",
    + "build-packages": "yarn turbo run packages:build",

    - "start": "concurrently --kill-others -n packages,app \"yarn build-packages:watch\" \"env-cmd env/.env.local
    + "app:start": "env-cmd env/.env.local yarn turbo run packages:build:watch app:start",

    - "build:test": "yarn build-packages && env-cmd env/.env.dev yarn --cwd packages/app build && mv packages/app/
    + "app:build:test": "env-cmd env/.env.dev yarn turbo run app:build && yarn run artifact:copy",

    - "bootstrap": "lerna bootstrap",
    - "build-packages:watch": "lerna run build:watch --parallel --scope package-1 --scope package-n ... 11 more ti

    + "artifact:copy": "rm -rf build && mv packages/app/build build",
  }
```

One more thing..

That's how the pipe looks like

```
    "turbo": {
      "baseBranch": "origin/master",
      "pipeline": {
        "packages:build": {
          "dependsOn": ["^packages:build"],
          "outputs": ["lib/**"]
        },
        "packages:build:watch": {
          "dependsOn": ["^packages:build:watch"],
          "outputs": ["lib/**"]
        },
        "app:build": {
          "dependsOn": ["packages:build"],
          "outputs": ["build/**"]
        },
        "app:start": {"cache": false},
        "test": {},
      }
    },
```