Summary

This documentation provide a high level view for the most used monorepo tools used today.

This documentation along with code structure-part-1-mono-repos-vs-poly-repo should introduce the reader to the monorepo subject and the supporting tools in order to better understand the context of these ADRs:

ADR	Name
ADR-0001	Mono repos over multi repo for FE
ADR-0002	Nrwl Nx as default tool to support our monorepo strategy
ADR-0003	Nrwl Nx CLI and Nx DevKit as default CLI and dev Kit
ADR-0004	Nx Ecosystem supporting tools
ADR-0005	Create our own plugins

What's Nx?

Nx is a **set of tools** built by the consulting firm Nrwl to help with exactly the issues of **consistency**, **safety**, and **maintainability** in the enterprise.

Nx includes a set of libraries, linters, and code generators to help large teams create and enforce best practices across their organizations.

Nx Workspace

A **workspace** is an example of a monorepo. Nx workspace is a repository that holds several related applications and libraries and keep them organized.

While we could keep each project in a separate repository, it would get messy **keeping all of the versions in sync**, **managing dependencies**, and ensuring that all developers working on the project have the **correct access they need**. Nx, a popular implementation of a monorepos solve these problems and more.

How Nx implement the monorepo features?

We will be presenting **Nx** implementation of these features along with **Lerna** implementation. Lerna is another monorepo that is more oriented towards npm packages, another way of sharing code between teams. We know that managing packages is more cumbersome than sharing code in the same repo. However, since not all projects could be added to a monorepo due to cross-organizational boundaries for instance.Even though Nx can manage/publish libraries, it may make sense to migrate reusable/standalone projects to an npm registry or another monorepo more suited for packages if things get bigger.

Quick disclaimer about terminology: The term "package" is what you can publish on the internet (for example, in the NPM - Node Package Manager). Lerna uses the name "project" as the wrapper folder of all the packages. Usually, a git repository is a "Lerna project" with several "Lerna packages". Nx, in the other hand, uses the name "project" to designate what Lerna calls "packages", and uses the name "workspace" to designate what Lerna calls "project".

Features we value most

Not all features are valued the same:

Category	Feature	Value	Nx	Lerna
Fast	Detecting affected projects	High	✔	❌
	Local computation caching	High	✔	❌
	Local task orchestration	High	✔	⚠
	Distributed computation caching	medium	✔	❌
	Distributed task execution	High	✔	❌
	Transparent remote execution	High	✔	✔
	Detecting affected projects	High	✔	✔
Understandable	Workspace analysis	High	✔	✔
	Dependency graph	High	✔	⚠
Manageable	Code sharing	High	✔	✔
	Consistent Tooling	High	✔	⚠
	Code generation	High	✔	⚠
	Project constraints and visibility	High	✔	⚠

Fast

- **Local computation caching**: The ability to store and replay file and process output of tasks. On the same machine, you will never build or test the same thing twice.

  ✔ **Nx**: Nx does tree diffing when restoring the results from its cache, which, on average, makes it faster than other tools.

  ❌ **Lerna**:Lerna doesn't support it and will always rerun everything from scratch.

- **Local task orchestration**: The ability to run tasks in the correct order and in parallel. All the listed tools can do it in about the same way, except Lerna, which is more limited.

  ✔ **Nx**: Supports it.

  ⚠ **Lerna**:Lerna's ability to do task coordination is more limited compared to the rest of the tools. It is not able to mix and match different targets (e.g., tests and builds), so it results in more idle time.

- **Distributed computation caching** : The ability to share cache artifacts across different environments. This means that your whole organization, including CI agents, will never build or test the same thing twice.

  ✓ **Nx**: Supports it.

  ❌ **Lerna**: Lerna cannot reuse computation across machines

- **Distributed task execution**:The ability to distribute a command across many machines, while largely preserving the dev ergonomics or running it on a single machine.

  ✓ **Nx**: Nx's implementation isn't as sophisticated as Bazel's but it can be turned on with a small configuration change.

  ❌ **Lerna**:Lerna doesn't support it.

- **Transparent remote execution**: The ability to execute any command on multiple machines while developing locally.

  ❌ **Nx**: Not supported.

  ❌ **Lerna**: Not supported.

- **Detecting affected projects/packages**:Determine what might be affected by a change, to run only build/test affected projects.

  ✓ **Nx**:Nx supports it. Its implementation doesn't just look at what files changed but also at the nature of the change.

  ✓ **Lerna**: Lerna supports it

Manageable

- **Code sharing**: Facilitates sharing of discrete pieces source code.

  ✓ **Nx**:Nx supports it. Any folder of files can be marked as a project and can be shared. Nx plugins help configure WebPack, Rollup, TypeScript and other tools to enable sharing without hurting dev ergonomics.

  ⚠️Lerna supports it. **Only npm packages can be shared**.

- **Consistent Tooling**: The tool helps you get a consistent experience regardless of what you use to develop your projects: different JavaScript frameworks (Angular, React),.Net, Rust, Java, etc. In other words, the tool treats different technologies the same way.

  ✓ **Nx**: Nx is pluggable. It is able to invoke npm scripts by default, but can be extended to invoke other tools (e.g., Gradle).

  ❌**Lerna**: Lerna can only run npm scripts.

- **Code generation**: Native support for generating code.

✔ **Nx** comes with powerful code generation capabilities. It uses a virtual file system and provides editor integration. Nx plugins provided generators for popular frameworks. Other generators can be used as well.

⚠**Lerna**:External generators can be used.

- **Project constraints and visibility**: Supports definition of rules to constrain dependency relationships within the repo. For instance, developers can mark some projects as private to their team so no one else can depend on them. Developers can also mark projects based on the technology used (e.g., React or Nest.js) and make sure that backend projects don't import frontend ones.

  ⚠**Lerna**: A linter with a set of custom rules and extra configuration can be used to ensure that some constraints hold.

  ✔ **Nx**: Developers can annotate projects in any way they seem fit, establish invariants, and Nx will make sure they hold. It allows developers to annotate what is private and what is not, what is experimental and what is stable, etc. Nx also allows you to define public API for each package, so other developers aren't able to deep import into them.

Understandable

- Workspace analysis: The ability to understand the project graph of the workspace without extra configuration.

  ✔ **Nx**:By default, Nx analyses package.json, JavaScript, and TypeScript files. It's pluggable and can be extended to support other platforms (e.g, Go, Java, Rust).

  ✔ **Lerna**:Lerna analyses package.json files.

- **Dependency graph**:Visualize dependency relationships between projects and/or tasks. The visualization is interactive meaning you are able to search, filter, hide, focus/highlight & query the nodes in the graph.visualization.

  ✔ **Nx**:Nx comes with an interactive visualizer that allows you to filter and explore large workspaces.

  ✔ **Lerna**:Lerna doesn't come with a visualizer but it's possible to write your own.

Nx Built-in tools:

Out of the box, Nx contains tools to help with:

- State management and NgRx
- Data persistence
- Code linting and formatting
- Migrating from AngularJS to Angular
- Analyzing dependencies visually
- Creating and running better tests
- Creating workspace-specific schematics

Categories for Libraries

To reduce the cognitive load, the Nx team recommends to categorize libraries as follows:

• **feature**: Implements a use case with smart components • **data-access**: Implements data accesses, e.g. via HTTP or WebSockets • **ui**: Provides use case-agnostic and thus reusable components (dumb components) • **util**: Provides helper functions Please note the separation between smart and dumb components. Smart components within feature libraries are use-case-specific.

More on Nx and Lerna:

Lerna	Nx	Description
Project	Workspace	collection of libraries/apps
Package	Project	Potentially packagable library

Although both are great tools to work with mono repo, they're quite different in their purpose.

Key differences

- Lerna is focused on linking multiple packages from the same project and managing npm publishing, and that's about it.
- Nx is more focused on managing development workflow for multiple projects. It means it can scaffold projects, and for every project, you can define configurations on how to run and build them, in a similar manner to Webpack.

Lerna fits better for **open source projects** with multiple packages (because you can easily publish your packages).

Nx **fits better** for handling complex workflows with multiple packages/projects.

How to choose

- If you don't intend to publish your packages, Nx might be a better fit.
- If you do intend to publish then, but you don't have a complex workflow, Lerna is definitely the way to go.
- If you want both (publish and complex workflow), neither one seems great, but you **should probably choose Nx**, and manage the publishing manually. Or, maybe, use Lerna and configure the workflow manually with Webpack.

Lerna is now maintained by Nrwl. This means that in the future there will be better integration between the two tools. **Stay tuned!**

Resources

[1] https://www.toptal.com/front-end/guide-to-monorepos#:~:text=A%20Monorepository%20is%20an%20architectural,in%20common%20with%20monolithic%20apps.

[2] https://lerna.js.org/

[3] https://monorepo.tools/#understanding-monorepos

[4] https://stackoverflow.com/questions/67000436/the-difference-between-nx-and-lerna-monorepos#:~:text=Lerna%20is%20focused%20on%20linking,development%20workflow%20for%20multiple%20packages.