In a folder

1. npm init –y
2. npm i typescript

(generates tsconfig.json, check the value target in this file . es2016 or es2020 or esnext)

1. npx tsc --init --rootdir src --outdir lib (source ts files in src and js files in outdir)
   1. npx tsc --init --rootdir src --outdir lib --sourceMap --declaration --declarationMap
2. compile ts files using npx tsc –watch. (This generates js files in lib (outdir) folder
3. run js files using node lib\<jsfile>

In a Node.js TypeScript context, the **tsconfig.json** file is a configuration file used by the TypeScript compiler (tsc) to define how TypeScript code should be compiled into JavaScript. It contains various settings and options to control how TypeScript behaves during compilation, such as specifying which files to include or exclude, setting the output directory, defining module resolution strategies, and more.

When you use TypeScript in a Node.js project, the tsconfig.json file helps configure the compilation process, making it easier to compile TypeScript code into JavaScript and integrate with your Node.js environment.



**Key Sections in tsconfig.json**

1. **compilerOptions**: This section contains all the important options for configuring the TypeScript compiler.
   * **target**: Specifies the JavaScript version to which TypeScript will compile (e.g., ES6, ES2020, ES5, etc.).
   * **module**: Defines the module system used for the compiled JavaScript (e.g., commonjs, es6, amd, etc.).
   * **outDir**: Specifies the directory where compiled JavaScript files will be placed (e.g., ./dist).
   * **rootDir**: Specifies the root directory where your TypeScript files are located (e.g., ./src).
   * **strict**: Enables strict type-checking options (e.g., strictNullChecks, noImplicitAny, etc.).
   * **esModuleInterop**: Allows default imports for CommonJS modules and provides better interoperability with modern JavaScript (e.g., import fs from 'fs';).
   * **skipLibCheck**: Skips type-checking of all declaration files (.d.ts).
   * **forceConsistentCasingInFileNames**: Ensures file names are consistent in casing across the project.
2. **include**: This section specifies which files or directories should be included in the TypeScript compilation. In the example above, it includes all .ts files inside the src directory.
3. **exclude**: This section specifies files or directories that should be excluded from the compilation process. *In the example above, it excludes the node\_modules directory,* which typically doesn't contain TypeScript files that need to be compiled.

Understanding the difference between **npm** and **npx** is important when working with **Node.js** and **JavaScript/TypeScript** projects.

**1. What is npm?**

**npm** stands for **Node Package Manager**. It is the default package manager for **Node.js** and is used for managing JavaScript packages (libraries, tools, frameworks, etc.) that you can install and use in your projects.

**npx** stands for **Node Package eXecute**. It is a tool that comes bundled with **npm** (starting from npm 5.2). **npx** allows you to **execute binaries** from Node modules (either globally or locally) without needing to install them globally or reference their paths manually.

**Key Differences: npm vs npx**

| **Feature** | **npm** | **npx** |
| --- | --- | --- |
| **Primary Purpose** | Package manager for installing, updating, and managing dependencies | Tool for executing commands without installing packages globally |
| **Usage** | Install dependencies (npm install), run scripts, etc. | Execute commands and binaries (e.g., npx create-react-app) |
| **Installation of packages** | Installs packages locally or globally | Does **not** install packages permanently; runs them temporarily |
| **Global Usage** | npm install -g <package-name> for global use | No need to install globally; runs packages directly |
| **Script Execution** | Can run scripts defined in package.json (npm run <script-name>) | Executes package binaries directly (npx <command>) |

**When to Use npm vs npx**

* **Use npm** when:
  + You need to install dependencies or packages into your project (local or global).
  + You are managing packages for a project (e.g., npm install to install dependencies from package.json).
  + You want to define and run custom scripts in your project (e.g., npm run start to run a custom script).
* **Use npx** when:
  + You want to **execute a command** from a package without installing it globally.
  + You need to **run binaries** that are installed locally within your project (e.g., npx is often used to run linters, generators, or test runners).
  + You want to **run a specific version** of a tool or package without worrying about installing it.

**Summary of Common TypeScript Commands**

| **Command** | **Description** |
| --- | --- |
| npx tsc --init | Initialize TypeScript configuration file (tsconfig.json). |
| npm install typescript --save-dev | Install TypeScript as a dev dependency in your project. |
| npx tsc | Compile TypeScript files based on tsconfig.json settings. |
| npx tsc --watch | Run TypeScript in **watch mode** to automatically recompile. |
| npx tsx index.ts | Run TypeScript code directly with tsx. |
| npx ts-node index.ts | Run TypeScript code directly with ts-node. |
| npm install @types/node --save-dev | Install Node.js type definitions for TypeScript. |
| npm install eslint --save-dev | Install ESLint for linting TypeScript code. |