Node.js

npm Basics

Introduction to Node.js

Node.js = JavaScript runtime built on **Chrome V8 engine**

- Lets you run JS **outside the browser**
- Ideal for:
  - Web servers and REST APIs
  - Command-line tools
  - Build and automation scripts

JVM & Node.js

You can think of Node.js as the JVM (Java Virtual Machine) equivalent for JavaScript:

- JVM runs compiled Java bytecode.
- Node.js runs JavaScript using the V8 engine (Google Chrome's JS engine).
- Both act as runtime environments — they provide memory management, libraries, and I/O handling that the language itself doesn't define.

Why Node.js?

- **Event-driven**, **non-blocking I/O**
- Handles many clients efficiently
- **Single-threaded**, but async
- Enables **JavaScript everywhere** (front + back)
- Supported by a vast **open-source ecosystem**

Many modern tools — including VS Code, Slack, Discord, and Postman — are built with JavaScript/Node.js using frameworks like Electron.

npm and npx

npm (Node Package Manager)

- Installs and manages packages
- Comes automatically with Node.js
- Creates and uses `package.json`

npx

- Runs packages **without global install**

```
npx cowsay "Hello!"
npx create-react-app myapp
```

package.json

Describes your Node.js project.

```
{
  "name": "myapp",
  "version": "1.0.0",
  "scripts": {
    "start": "node index.js"
  },
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

npm install — Quick Reference

Purpose	Command	Description
Install all dependencies	`npm install`	Installs everything listed in **package.json**
Add a new package	`npm install express`	Installs a package and saves it to **dependencies**
Add a dev-only tool	`npm install --save-dev nodemon`	Installs a package only for development (not production)
Remove a package	`npm uninstall express`	Removes the package and updates **package.json**
Reinstall from scratch	`rm -rf node_modules && npm install`	Cleans and reinstalls dependencies

> Tip: Use `-g` to install globally (for CLI tools like `nodemon` or `eslint`).

npm run scripts

Use "scripts" section to define commands.

```
"scripts": {
  "start": "node index.js",
  "dev": "nodemon index.js",
  "test": "echo 'Running tests...'"
}
```

Run with:

```
npm run dev
npm start   # shortcut for "run start"
```

Node.js Development Utilitaies

Using dotenv

Manage environment variables safely.

Install: `npm install dotenv`

Create a .env file:

```
PORT=3000
API_KEY=abcd1234
```

Use it in your code:

```
require('dotenv').config();
console.log(process.env.PORT);
```

Useful Dev Tools

Tool	Purpose	Install Command
nodemon	Auto-restart server on file changes	`npm i -D nodemon`
eslint	Enforce code style & catch errors	`npm i -D eslint`
prettier	Auto-format code	`npm i -D prettier`
dotenv	Manage environment variables	`npm i dotenv`
jest	Testing framework	`npm i -D jest`

-D option

Use `-D` or `--save-dev` to mark a package as **development-only** (not needed in production).

When you install packages using:

```
npm install --save-dev <package>
# or shorter:
npm i -D <package>
```

It means the package is only needed during development, not when your app runs in production.

Example

```
npm i –D nodemon eslint prettier
npm i express dotenv
```

Type	Example Packages	Purpose
Dependencies	express, dotenv	Required for the app to **run** in production
Dev Dependencies	nodemon, eslint, prettier	Used only during **development** (not needed in production)

When deploying your app, only **dependencies** are installed with `npm install --production`.

- Only runtime packages (under "dependencies") are installed.
- Development tools like nodemon, eslint, prettier, and jest are not installed.
- Your node_modules folder becomes smaller and lighter ideal for deployment to servers or Docker containers.