Click Here

Node.js is a JavaScript runtime for server-side programming. It allows developers to create scalable backend functionality using JavaScript, a language many are already familiar with from browser-based web development. In this guide, we will show you three different ways of getting Node.js installed on an Ubuntu server: using apt to install the nodejs package from Ubuntu’s default software repository using apt with an alternate PPA software repository to install specific versions of the nodejs package installing nvm, the Node Version Manager, and using it to install and manage multiple versions of Node.js For many users, using apt with the default repo will be sufficient. If you need specific newer or legacy versions of Node, you should use the PPA repository. If you are actively developing Node applications and need to switch between node versions frequently, choose the nvm method. Note This article will walk you through installing Node.js on an Ubuntu server. If you wanted a 1-click way to deploy a Node application to a live server, take a look at DigitalOcean App Platform. To follow this guide, you will need an Ubuntu 20.04 server set up. Before you begin, you should have a non-root user account with sudo privileges set up on your system. You can learn how to do this by following the Ubuntu initial server setup tutorial. Ubuntu 20.04 contains a version of Node.js in its default repositories that can be used to provide a consistent experience across multiple systems. At the time of writing, the version in the repositories is 10.19. This will not be the latest version, but it should be stable and sufficient for quick experimentation with the language. Warning: the version of Node.js included with Ubuntu 20.04, version 10.19, is now unsupported and unmaintained. You should not use this version in production, and should refer to one of the other sections in this tutorial to install a more recent version of Node. To get this version, you can use the apt package manager. Refresh your local package index first: Then install Node.js: Check that the install was successful by querying node for its version number: Outputv10.19.0 If the package in the repositories suits your needs, this is all you need to do to get set up with Node.js. In most cases, you’ll also want to also install npm, the Node.js package manager. You can do this by installing the npm package with apt: This allows you to install modules and packages to use with Node.js. At this point, you have successfully installed Node.js and npm using apt and the default Ubuntu software repositories. The next section will show how to use an alternate repository to install different versions of Node.js. To install a different version of Node.js, you can use a PPA (personal package archive) maintained by NodeSource. These PPAs have more versions of Node.js available than the official Ubuntu repositories. Node.js v16 and v18 are available as of the time of writing. First, install the PPA to get access to its packages. From your home directory, use curl to retrieve the installation script for your preferred version, making sure to replace 16.x with your preferred version string (if different): cd ~ curl -sL -o /tmp/nodesource_setup.sh Refer to the NodeSource documentation for more information on the available versions. Inspect the contents of the downloaded script with nano or your preferred text editor: nano /tmp/nodesource_setup.sh When you are satisfied that the script is safe to run, exit your editor. Then run the script with sudo: sudo bash /tmp/nodesource_setup.sh The PPA will be added to your configuration and your local package cache will be updated automatically. You can now install the Node.js package in the same way you did in the previous section: Verify that you’ve installed the new version by running node with the -v version flag: Outputv16.19.0 The NodeSource nodejs package contains both the node binary and npm, so you don’t need to install npm separately. At this point, you have successfully installed Node.js and npm using apt and the NodeSource PPA. The next section will show how to use the Node Version Manager to install and manage multiple versions of Node.js. Another way of installing Node.js that is particularly flexible is to use nvm, the Node Version Manager. This piece of software allows you to install and maintain many different independent versions of Node.js, and their associated Node packages, at the same time. To install NVM on your Ubuntu 20.04 machine, visit the project’s GitHub page. Copy the curl command from the README file that displays on the main page. This will get you the most recent version of the installation script. Before piping the command through to bash, it is always a good idea to audit the script to make sure it isn’t doing anything you don’t agree with. You can do that by removing the | bash segment at the end of the curl command: curl -o- Review the script and make sure you are comfortable with the changes it is making. When you are satisfied, run the command again with | bash appended at the end. The URL you use will change depending on the latest version of nvm, but as of right now, the script can be downloaded and executed with the following: curl -o- | bash This will install the nvm script to your user account. To use it, you must first source your .bashrc file: Now, you can ask NVM which versions of Node are available: Output. . . v18.0.0 v18.1.0 v18.2.0 v18.3.0 v18.4.0 v18.5.0 v18.6.0 v18.7.0 v18.8.0 v18.9.0 v18.9.1 v18.10.0 v18.11.0 v18.12.0 (LTS: Hydrogen) v18.12.1 (LTS: Hydrogen) v18.13.0 (Latest LTS: Hydrogen) v19.0.0 v19.0.1 v19.1.0 v19.2.0 v19.3.0 v19.4.0 It’s a very long list. You can install a version of Node by writing in any of the release versions listed. For instance, to get version v14.10.0, you can run: You can view the different versions you have installed by listing them: Output-> v14.10.0 v14.21.2 default -> v14.10.0 iojs -> N/A (default) unstable -> N/A (default) node -> stable (-> v14.21.2) (default) stable -> 14.21 (-> v14.21.2) (default) . . . This shows the currently active version on the first line (-> v14.10.0), followed by some named aliases and the versions that those aliases point to. Note: if you also have a version of Node.js installed through apt, you may receive a system entry here. You can always activate the system-installed version of Node using nvm use system. Additionally, there are aliases for the various long-term support (or LTS) releases of Node: Outputlts/* -> lts/hydrogen (-> N/A) lts/argon -> v4.9.1 (-> N/A) lts/boron -> v6.17.1 (-> N/A) lts/carbon -> v8.17.0 (-> N/A) lts/dubnium -> v10.24.1 (-> N/A) lts/erbium -> v12.22.12 (-> N/A) lts/fermium -> v14.21.2 lts/gallium -> v16.19.0 (-> N/A) lts/hydrogen -> v18.13.0 (-> N/A) You can install a release based on these aliases as well. For instance, to install the latest long-term support version, hydrogen, run the following: OutputDownloading and installing node v18.13.0... . . . Now using node v18.13.0 (npm v8.19.3) You can switch between installed versions with nvm use: OutputNow using node v14.10.0 (npm v6.14.8) You can verify that the install was successful using the same technique from the other sections: node -v v14.10.0 The correct version of Node is installed on your machine as expected. A compatible version of npm is also available. You can uninstall Node.js using apt or nvm, depending on how it was installed. To remove the version from the system repositories, use apt remove: By default, apt remove retains any local configuration files that were created since installation. If you don’t want to save the configuration files for later use, use apt purge: To uninstall a version of Node.js that you installed using nvm, first determine whether it is the current active version: If the version you are targeting is not the current active version, you can run: nvm uninstall node_version OutputUninstalled node node_version This command will uninstall the selected version of Node.js. If the version you would like to remove is the current active version, you first need to deactivate nvm to enable your changes: Now you can uninstall the current version using the uninstall command used previously. This removes all files associated with the targeted version of Node.js. When starting a new project, installing Node.js is a crucial step. Node.js allows developers to run JavaScript on the server-side, making it a popular choice for web applications. For example, if you’re building a real-time chat application, you might want to use Node.js with a framework like Express.js to handle incoming requests and send responses. To illustrate this, let’s consider a scenario where you’re building a simple chat application using Node.js and Express.js. You want to create a server that listens for incoming messages and broadcasts them to all connected clients. Here’s a simple example of how you might structure your server-side code using Node.js and Express.js: const express = require('express'); const app = express(); const server = require('http').createServer(app); const io = require('socket.io')(server); app.get('/', (req, res) => { res.sendFile(__dirname + '/index.html'); }); io.on('connection', (socket) => { console.log('a user connected'); socket.on('disconnect', () => { console.log('user disconnected'); }); socket.on('chat message', (msg) => { io.emit('chat message', msg); }); }); server.listen(3000, () => { console.log('listening on *:3000'); }); This example demonstrates how Node.js can be used to create a server that handles real-time communication between clients. To install Node.js for app development, you can use the methods described earlier in this tutorial. For instance, you can use apt to install Node.js from the Ubuntu repository or use nvm to install a specific version of Node.js. Here’s an example command to install Node.js using apt: sudo apt update && sudo apt install nodejs Alternatively, you can use nvm to install a specific version of Node.js. First, install nvm if you haven’t already, then use the following command to install a specific version of Node.js: nvm install Replace with the version number you want to install. Setting up Backend Projects Node.js is widely used for setting up backend projects due to its event-driven, non-blocking I/O model. This makes it lightweight and efficient for handling multiple concurrent connections. For example, if you’re building a RESTful API, Node.js can be used to create the server-side logic, handle database interactions, and return data to the client. When setting up a backend project, you might want to use a framework like Express.js or Koa.js to simplify the process. These frameworks provide a structure for organizing your code and handling HTTP requests and responses. Managing Multiple Node.js Versions with NVM Managing multiple Node.js versions is essential when working on different projects that require different versions of Node.js. NVM (Node Version Manager) is a popular tool for managing multiple Node.js versions on a single machine. For example, let’s say you’re working on two projects: one that requires Node.js version 14.x and another that requires Node.js version 16.x. With NVM, you can easily switch between these versions without affecting the system’s default Node.js version. To switch between Node.js versions using NVM, you can use the following command: nvm use Replace with the version number you want to use. This command will switch your Node.js version to the specified one, allowing you to work on your project without affecting other projects that require different versions. Permission issues with global installs occur when you try to install a package globally using npm without having the necessary permissions. This is because global packages are installed in a system directory that requires root access. Why it occurs: This error occurs because npm tries to write to a directory that requires root access, but the user running npm does not have those permissions. How to fix it: To fix this issue, you can use one of the following methods: Run the command with sudo: sudo npm install -g Change the ownership of the npm directory to the current user: sudo chown -R $USER:$(whoami) /usr/local/{lib/node_modules,bin,share/man} && sudo chown -R $USER:$(whoami) ~/.npm Use a package manager like nvm or yarn that does not require root access for global installs. NVM not found after install (profile not sourced) NVM not found after install occurs when you install NVM but do not source the NVM script in your shell profile. Why it occurs: This error occurs because NVM is not added to your system’s PATH after installation. The NVM script needs to be sourced in your shell profile to make NVM available. How to fix it: To fix this issue, you need to add the following line to your shell profile file (e.g., ~/.bashrc or ~/.zshrc): source ~/.nvm/nvm.sh Then, restart your terminal or run source ~/.nvm/nvm.sh to apply the changes. Conflicts between apt and NVM-installed versions Conflicts between apt and NVM-installed versions occur when you have Node.js installed using both apt and NVM. This can cause issues with package management and version conflicts. Why it occurs: This error occurs because both apt and NVM install Node.js in different locations, leading to conflicts between the two installations. How to fix it: To fix this issue, you should remove the Node.js installation from apt and use NVM as the primary method for managing Node.js versions. Here’s how to do it: Remove Node.js installed using apt: sudo apt purge nodejs Install Node.js using NVM: nvm install node Make sure to use NVM to manage your Node.js versions going forward. Method Description Pros Cons Example Apt Installs Node.js from Ubuntu’s repository Easy to install, no additional setup required Limited version control, may not be the latest version sudo apt install nodejs NVM Installs and manages multiple Node.js versions Allows for easy version switching, supports multiple versions Requires additional setup, can be complex for beginners nvm install node NodeSource PPA Installs Node.js from a third-party repository Provides access to the latest Node.js versions, easy to install May not be as secure as other methods, requires adding a third-party repository sudo apt update && sudo apt install nodejs Binary Installs Node.js from pre-compiled binaries Fast installation, no dependencies required May not be compatible with all systems, requires manual updates sudo apt update && sudo apt install nodejs Source Installs Node.js from source code Provides complete control over the installation, can be customized Time-consuming, requires technical expertise, manual updates required sudo apt update && sudo apt install nodejs Note: The examples provided are simplified and may require additional steps or configurations depending on the specific installation method and system requirements. The best way to install Node.js on Ubuntu 20.04 is to use a package manager like NVM (Node Version Manager) or a NodeSource PPA. These methods provide more flexibility and allow for easy version management. Example: # Install Node.js using NVM nvm install node # Install Node.js using NodeSource PPA sudo apt update && sudo apt install nodejs To check your Node.js version, run the command node -v in your terminal. This will display the version of Node.js installed on your system. Example: node -v Yes, you can install multiple versions of Node.js using a package manager like NVM. NVM allows you to install, manage, and switch between different versions of Node.js easily. Example: # Install a specific version of Node.js using NVM nvm install 14.x # List all installed Node.js versions nvm ls NVM (Node Version Manager) is a tool that allows you to easily install, manage, and switch between different versions of Node.js on your system. You should use NVM because it provides a convenient way to manage multiple Node.js versions, which is particularly useful for developers who need to work with different projects that require different Node.js versions. Example: # Install NVM curl -o- | bash # Use NVM to install Node.js nvm install node To uninstall Node.js completely, you need to remove all the Node.js packages and dependencies. If you installed Node.js using apt, you can uninstall it by running sudo apt purge nodejs. If you used NVM, you can uninstall Node.js by running nvm uninstall node. Example: # Uninstall Node.js using apt sudo apt purge nodejs # Uninstall Node.js using NVM nvm uninstall node No, Ubuntu 20.04 does not come with Node.js preinstalled. You need to install it manually using a package manager or by downloading the binaries. Example: # Check if Node.js is installed node -v To update Node.js on Ubuntu, you can use the package manager you used to install it. If you used apt, you can update Node.js by running sudo apt update && sudo apt full-upgrade. If you used NVM, you can update Node.js by running nvm install node --latest. Example: # Update Node.js using apt sudo apt update && sudo apt full-upgrade # Update Node.js using NVM nvm install node --latest There are several ways to set up Node.js on your Ubuntu server, each with its own advantages. Whether you choose to use the packaged version, nvm, or a NodeSource PPA, the key is to select the method that best fits your project’s requirements. To further explore Node.js and its applications, we recommend checking out the following tutorials and resources: By following these guides, you’ll be well on your way to mastering Node.js and deploying your applications efficiently. DSA to Development: A Complete GuideBeginner to AdvanceJAVA Backend Development - LiveIntermediate and AdvanceTech Interview 101 - From DSA to System Design for Working ProfessionalsBeginner to AdvanceFull Stack Development with React & Node JS - LiveBeginner to AdvanceC++ Programming Course Online - Complete Beginner to AdvancedBeginner to AdvanceJava Programming Online Course [Complete Beginner to Advanced]Beginner to AdvancePage 2Our website uses cookiesWe use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy Policy Node.js is an open-source, cross-platform JavaScript runtime environment. For information on using Node.js, see the Node.js website. The Node.js project uses an open governance model. The OpenJS Foundation provides support for the project. Contributors are expected to act in a collaborative manner to move the project forward. We encourage the constructive exchange of contrary opinions and compromise. The TSC reserves the right to limit or block contributors who repeatedly act in ways that discourage, exhaust, or otherwise negatively affect other participants. This project has a Code of Conduct. Looking for help? Check out the instructions for getting support. Current: Under active development. Code for the Current release is in the branch for its major version number (for example, v22.x). Node.js releases a new major version every 6 months, allowing for breaking changes. This happens in April and October every year. Releases appearing each October have a support life of 8 months. Releases appearing each April convert to LTS (see below) each October. LTS: Releases that receive Long Term Support, with a focus on stability and security. Every even-numbered major version will become an LTS release. LTS releases receive 12 months of Active LTS support and a further 18 months of Maintenance. LTS release lines have alphabetically-ordered code names, beginning with v4 Argon. There are no breaking changes or feature additions, except in some special circumstances. Nightly: Code from the Current branch built every 24-hours when there are changes. Use with caution. Current and LTS releases follow semantic versioning. A member of the Release Team signs each Current and LTS release. For more information, see the Release README. Binaries, installers, and source tarballs are available at . The latest directory is an alias for the latest Current release. The latest-codename directory is an alias for the latest release from an LTS line. For example, the latest-hydrogen directory contains the latest Hydrogen (Node.js 18) release. Each directory and filename includes the version (e.g., v22.0.0), followed by the UTC date (e.g., 20240424 for April 24, 2024), and the short commit SHA of the HEAD of the release (e.g., ddd0a9e494). For instance, a full directory name might look like v22.0.0-nightly20240424ddd0a9e494. Documentation for the latest Current release is at . Version-specific documentation is available in each release directory in the docs subdirectory. Version-specific documentation is also at . Download directories contain a SHASUMS256.txt file with SHA checksums for the files. To download SHASUMS256.txt using curl: curl -O To check that downloaded files match the checksum, use sha256sum: sha256sum -c SHASUMS256.txt --ignore-missing For Current and LTS, the GPG detached signature of SHASUMS256.txt is in SHASUMS256.txt.sig. You can use it with gpg to verify the integrity of SHASUMS256.txt. You will first need to import the GPG keys of individuals authorized to create releases. See Release keys for commands to import active release keys. Next, download the SHASUMS256.txt.sig for the release: curl -O Then use gpg --verify SHASUMS256.txt.sig SHASUMS256.txt to verify the file's signature. See BUILDING.md for instructions on how to build Node.js from source and a list of supported platforms. For information on reporting security vulnerabilities in Node.js, see SECURITY.md. For information about the governance of the Node.js project, see GOVERNANCE.md. TSC emeriti members Emeriti Collaborators follow the Collaborator Guide in maintaining the Node.js project. Triagers follow the Triage Guide when responding to new issues. Primary GPG keys for Node.js Releasers (some Releasers sign with subkeys): To import the full set of trusted release keys (including subkeys possibly used to sign releases): gpg --keyserver hkps://keys.openpgp.org --recv-keys C0D6248439F1D5604AAFFB4021D900FFDB233756 # Antoine du Hamel gpg --keyserver hkps://keys.openpgp.org --recv-keys DD792F5973C6DE52C432CBDAC77ABFA00DDBF2B7 # Juan José Arboleda gpg --keyserver hkps://keys.openpgp.org --recv-keys CC68F5A3106FF448322E48ED27F5E38D5B0A215F # Marco Ippolito gpg --keyserver hkps://keys.openpgp.org --recv-keys 8FCCA13FEF1D0C2E91008E09770F7A9A5AE15600 # Michaël Zasso gpg --keyserver hkps://keys.openpgp.org --recv-keys 890C08DB8579162FEE0DF9DB8BEAB4DFCF555EF4 # Rafael Gonzaga gpg --keyserver hkps://keys.openpgp.org --recv-keys C82FA3AE1CBEDC6BE46B9360C43CEC45C17AB93C # Richard Lau gpg --keyserver hkps://keys.openpgp.org --recv-keys 108F52B48DB57BB0CC439B2997B01419BD92F80A # Ruy Adorno gpg --keyserver hkps://keys.openpgp.org --recv-keys A363A499291CBBC940DD62E41F10027AF002F8B0 # Ulises Gascón See Verifying binaries for how to use these keys to verify a downloaded file. Other keys used to sign some previous releases When possible, the commitment to take slots in the security release steward rotation is made by companies in order to ensure individuals who act as security stewards have the support and recognition from their employer to be able to prioritize security releases. Security release stewards manage security releases on a rotation basis as outlined in the security release process. Node.js is available under the MIT License. Node.js also includes external libraries that are available under a variety of licenses. See LICENSE for the full license text. A NodeJS web server is a server built using NodeJS to handle HTTP requests and responses. Unlike traditional web servers like Apache or Nginx, which are primarily designed to give static content, NodeJS web servers can handle both static and dynamic content while supporting real-time communication. It uses JavaScript (the same language used for client-side scripting) on the server side, making it a popular choice for full-stack developers. Setting Up a NodeJS Web ServerStep 1: Install NodeJSIf you haven't installed NodeJS in your system, follow the article- How to Install NodeJSTo verify the installation, open your terminal or command prompt and type:node -vThis will display the installed NodeJS version.Step 2: Create Your Project DirectoryCreate a new directory for your project and navigate into it:mkdir node-server cd node-serverStep 3: Initialize the ProjectCreate a package.json file, which contains metadata about your project:npm init -yStep 4: Create a Basic ServerCreate a file named server.jsOpen server.js in your code editor and add the following code to create a simple HTTP server: JavaScript const http = require('http'); const server = http.createServer((req, res) => { res.writeHead(200, { 'Content-Type': 'text/plain' }); res.end('Hello, World!'); }); const port = 3000; const host = 'localhost'; server.listen(port, host, () => { console.log(`Server running at http://${host}:${port}/`); }); OutputRun the server by using the below commandnode server.jsWeb ServerIn this examplehttp.createServer(): Creates an HTTP server that listens for requests.res.writeHead(): Sends a response header with the status code 200 (OK).res.end(): Ends the response and sends the message "Hello, World!" to the client.server.listen(): Starts the server on the specified host (localhost) and port (3000).Why Use NodeJS for Web Servers?1. High PerformanceNodeJS is designed for speed. Its non-blocking I/O model and the V8 engine allow for fast execution of JavaScript code, making it ideal for handling high-concurrency scenarios, such as APIs or real-time applications.2. Single Language StackNodeJS enables developers to use JavaScript both on the client side and server side. This eliminates the need to switch between different programming languages and allows for better integration between the front-end and back-end.3. ScalabilityNodeJS is scalable by design due to its event-driven architecture. It can easily handle more requests with minimal overhead by adding more processes or utilizing a load balancer.4. Real-Time DataNodeJS is especially well-suited for applications that require real-time data, such as chat applications, live notifications, and collaborative platforms. Its WebSocket support allows for full-duplex communication between the client and server.5. Large EcosystemNodeJS has an extensive ecosystem with over a million open-source libraries available through npm. These libraries make it easy to add functionality such as authentication, routing, data handling, and much more.How Does a NodeJS Web Server Work?Event Loop: The server listens for incoming requests and processes them in a non-blocking way. When a request comes in, it is passed to the event loop, which decides how to handle it.Non-Blocking I/O: While the server is processing a request (like querying a database or reading from a file), it does not block other incoming requests. This is handled asynchronously through callbacks, promises, or async/await syntax.Request and Response: Once the event loop processes the request, the server sends a response back to the client (browser, API consumer, etc.). The response could be in HTML, JSON, or other data format.Web Servers: NodeJS can serve static files (like images or stylesheets) and dynamic content (like API responses) using built-in modules like http, fs, and path.Routing in NodeJS Web ServerIn real-world applications, your server needs to handle different routes (URLs) and respond differently based on the request. Let us understand with the help of the example: JavaScript // server.js const http = require('http'); const hostname = '127.0.0.1'; const port = 3000; const server = http.createServer((req, res) => { if (req.url === '/') { res.statusCode = 200; res.setHeader('Content-Type', 'text/plain'); res.end('Welcome to the Homepage!'); } else if (req.url === '/about') { res.statusCode = 200; res.setHeader('Content-Type', 'text/plain'); res.end('This is the About Page'); } else if (req.url === '/contact') { res.statusCode = 200; res.setHeader('Content-Type', 'text/plain'); res.end('This is the Contact Us Page'); } else { res.statusCode = 404; res.setHeader('Content-Type', 'text/plain'); res.end('Page Not Found'); } }); server.listen(port, hostname, () => { console.log(`Server running at http://${hostname}:${port}/`); }); OutputIn this exampleThe http.createServer() method initializes the server.res.writeHead() sends a response header with the status code (e.g., 200 OK) and any other headers.res.statusCode sets the HTTP status code for the response. In this case, 200 means the request was successful.res.setHeader() sets the content type of the response. In this case, text/plain means the server is sending plain text back to the client.server.listen() binds the server to the specified port and hostname (in this case, localhost:3000).When the server successfully starts, the callback function is executed, logging a message (Server running at to the console.Key Benefits of Using a NodeJS Web ServerReal-Time Capabilities: NodeJS can be used in the real-time applications like chat apps or live data streaming due to its non-blocking architecture and event-driven model.High Performance: The V8 engine provides fast execution of JavaScript, making NodeJS a great choice for applications requiring high performance and low latency.Scalability: NodeJS is highly scalable, thanks to its event loop and non-blocking I/O. It can handle a large number of simultaneous requests efficiently.Lightweight and Efficient: With its single-threaded, non-blocking nature, NodeJS reduces the overhead of traditional multi-threaded servers, making it lightweight and ideal for handling concurrent requests. Node.js Tutorial NodeJS Introduction Node.js Roadmap: A Complete Guide How to Install Node.js on Linux How to Install Node.js on Windows How to Install NodeJS on MacOS Node.js vs Browser - Top Differences That Every Developer Should Know NodeJS REPL (READ, EVAL, PRINT, LOOP) Explain V8 engine in Node.js Node.js Web Application Architecture NodeJS Event Loop NodeJS Modules What are Buffers in Node.js ? Node.js Streams NodeJS NPM Steps to Create and Publish NPM packages Introduction to NPM scripts Node.js package.json What is package-lock.json ? Node Debugging How to Perform Testing in Node.js ? Unit Testing of Node.js Application NODE_ENV Variables and How to Use Them ? Difference Between Development and Production in Node.js Best Security Practices in Node.js Deploying Node.js Applications How to Build a Microservices Architecture with NodeJS Node.js with WebAssembly Node.js Web Server Node Exercises, Practice Questions and Solutions Node.js Projects NodeJS Interview Questions and Answers Uploading images is a crucial feature in web development, especially for profile pictures or when handling user-submitted content like resumes. In this guide, we’ll explore how to seamlessly upload and display images with additional form data using Node.js and React.js.Backend SetupLet’s begin with the backend setup, where we’ll use Express.js for building RESTful APIs, Sequelize as the ORM for MySQL, Multer for handling image uploads, and CORS to manage cross-origin resource sharing.npm install express sequelize mysql2 multer corsLet start,we start from dbconfig to configure database.Here is dbconfig code.Database Configuration (dbconfig.js)// dbconfig.jsconst { Sequelize } = require("sequelize");const sequelize = new Sequelize("image", "root", "", { host: "localhost", dialect: "mysql",});(async () => { try { await sequelize.authenticate(); console.log("Connection has been established successfully."); } catch (error) { console.error("Unable to connect to the database:", error); } })();(async () => { await sequelize.sync(); console.log("All models were synchronized successfully");})();module.exports = sequelize;Model Definition (image.js)//image.jsconst sequelize = require("../database/dbconfig");const { DataTypes } = require("sequelize");const Image = sequelize.define("image", { name: { type: DataTypes.STRING, allowNull: false, }, gender: { type: DataTypes.STRING, allowNull: false, }, image: { type: DataTypes.STRING, allowNull: true, }, }, { timestamps: false, });module.exports = Image;so we can that we have add three column in our database Name,gender and imageUrl.next we will see our index.js file.Server Configuration (index.js)//index.jsconst express = require("express");const imageRouter = require("./api/routes/image");const cors = require("cors");const app = express();app.use(express.json());app.use(express.urlencoded({ extended: true }));app.use(express.static("public"));app.use(cors());app.use("/", imageRouter);app.listen(8080, () => { console.log("server is running on port 8080");});so to get json data we use express.json() and our public folder is static folder/directory. where we save our images.and we run server on port 8080.next we will see our controller file where we write our function to handle request and responseController (image.js)const Image = require("../model/image");exports.getUser = async (req, res) => { try { const user = await Image.findAll(); res.status(200).json(user); } catch (error) { res.status(500).json({ error }); }};exports.uploadImage = async (req, res) => { try { const image = req.file ? req.file.filename : null; const { name, gender } = req.body; if (!req.file) { res.status(400).send("no file uploaded"); } let user = await Image.create({ name, gender, image }); res.status(201).json({ user, message: "data upload successfully" }); } catch (error) { res.status(500).json({ error, message: "internal server error " }); }Our first function is to get all data that are saved in our database and in second function we post data and imageURl to databaseNow we define our route and config multer as follow.const { uploadImage, getUser } = require("../controller/image");const express = require("express");const multer = require("multer");const storage = multer.diskStorage({ destination: function (req, file, cb) { return cb(null, "./public/images"); }, filename: function (req, file, cb) { return cb(null, Date.now() + "-" + file.originalname); },});const upload = multer({ storage: storage });const router = express.Router();router.get("/", getUser);router.post("/upload", upload.single("image"), uploadImage);module.exports = router;so thats all for backend we can test only backend using postman api but we will upload image from frontend using React and save image on backend server and save image url on backend.Frontend Development with React.jsnow lets take a look for our frontend code.Install Reactnpx create-react-app myappnow we open it in vscode and go to “src” folder and open app.js file to remove all code that are in that file and write this code.import React from "react";import axios from "axios";function App() { const [formData, setFormData] = React.useState({ name: "", gender: "", image: "", }); const [displayData, setDisplayData] = React.useState([]); const handleChange = (event) => { const { value, name } = event.target; setFormData({ ...formData, [name]: value }); }; const handleFileChange = (event) => { const file = event.target.files[0]; console.log(file); setFormData({ ...formData, image: file }); }; const handleSubmit = (event) => { event.preventDefault(); axios .post(" , formData, { headers: { "Content-Type": "multipart/form-data", // Set content type for FormData }, }) .catch((error) => { console.log("Error in post data" + error.message); }); setFormData({ name: "", gender: "", image: "", }); console.log("successfully created"); }; const shortPoll = React.useEffect(() => { (async () => { await axios.get(" ").then((res) => { setDisplayData(res.data); console.log(res.data, "res.data"); }); })(); }, []); console.log(shortPoll); setInterval(() => { console.log(1); // shortPoll(); }, 2000); // console.log(data); return (Male Female Submit name genger image {displayData.map(item) => { return ({item.name} {item.gender}); }}});}export default App;now we write all our code. To get image from backend we write domain where our server is running and write folder name where we have saved our images and then filename from database.This comprehensive guide takes you through setting up the backend with Node.js, Sequelize, and Multer, and creating a React frontend to seamlessly upload and display images with additional form data. Feel free to explore and customize this solution for your specific use case! In this tutorial, I will show you how to upload/store images in MongoDB using Node.js & Express with the help of multer & multer-gridfs-storage. Related Posts: – How to upload multiple files in Node.js – Upload & resize multiple images in Node.js using Express, Multer, Sharp – Node.js Express File Upload Rest API (static folder) example using Multer – Google Cloud Storage with Node.js: File Upload example More Practice: – Node.js, Express & MongoDb: Build a CRUD Rest Api example – Node.js + MongoDB: User Authentication & Authorization with JWT Deployment: Docker Compose: Node.js Express and MongoDB example Overview Our Node.js Application will provide APIs for: uploading Files/Images to MongoDB getting list of Files’ information (file name & url) downloading File/Image from server with the url This is the UI: You can use an HTTP Client for image upload: The images will be stored in 2 collections: photo.files and photos.chunks. If we get list of image files, the Node.js Rest Apis will return: Each item in the response array has a url that you can use for downloading the file/image. These are APIs to be exported: Methods Urls Actions POST /upload upload a File/Image GET /files get List of Images (name & url) GET /files/[filename] download a File This Node.js App works with: – Angular 8 / Angular 10 / Angular 11 / Angular 12 / Angular 13 / Angular 14 / Angular 15 / Angular 16 / Angular 17 – Angular Material 12 / Angular Material 14 / Angular Material 15 / Angular Material 16 / Angular Material 17 – Vue Client / Vuetify Client – React Client / React Hooks Client – React Image Upload with Preview – Material UI Client – Axios Client Node.js upload/store image in MongoDB We’re gonna show you how to build this Node.js app step by step. Project Structure Now look at our project structure: – config/db.js: includes configuration for MongoDB and Multer (url, database, image Bucket). – views/index.html: contains HTML form for user to upload images. – routes/index.js: defines routes for endpoints that is called from views, use controllers to handle requests. – controllers: home.js returns views/index.html upload.js handles upload, store, display and download images – middleware/upload.js: initializes Multer GridFs Storage engine (including MongoDB) and defines middleware function. – server.js: initializes routes, configure CORS, runs Express app. Setup Node.js modules Open command prompt, change current directory to the root folder of our project. Install Express, CORS, Multer, Multer GridFs Storage and MongoDB with the following command: npm install express cors multer multer-gridfs-storage mongodb The package.json file will look like this: { "name": "upload-multiple-images-mongodb", "version": "1.0.0", "description": "Node.js upload multiple files/images to MongoDB Demo", "main": "src/server.js", "scripts": { "test": "echo \"Error: no test specified\" && exit 1" }, "keywords": ["node", "upload", "multiple", "files", "images", "mongodb"], "author": "bezkoder", "license": "ISC", "dependencies": { "cors": "^2.8.5", "express": "^4.17.1", "mongodb": "^4.1.3", "multer": "^1.4.3", "multer-gridfs-storage": "^5.0.2" } } Create View for uploading image In views folder, create index.html file with the HTML and Javascript code as below: Node.js upload images { try { await upload(req, res); console.log(req.file); if (req.file == undefined) { return res.send({ message: "You must select a file.", }); } return res.send({ message: "File has been uploaded.", }); } catch (error) { console.log(error); return res.send({ message: "Error when trying upload image: ${error}", }); } }; const getListFiles = async (req, res) => { try { await mongoClient.connect(); const database = mongoClient.db(dbConfig.database); const images = database.collection(dbConfig.imgBucket + ".files"); const cursor = images.find({}); if ((await cursor.count()) === 0) { return res.status(500).send({ message: "No files found!", }); } let fileInfos = []; await cursor.forEach((doc) => { fileInfos.push({ name: doc.filename, url: baseUrl + doc.filename, }); }); return res.status(200).send(fileInfos); } catch (error) { return res.status(500).send({ message: error.message, }); } }; const download = async (req, res) => { try { await mongoClient.connect(); const database = mongoClient.db(dbConfig.database); const bucket = new GridFSBucket(database, { bucketName: dbConfig.imgBucket, }); let downloadStream = bucket.openDownloadStreamByName(req.params.name); downloadStream.on("data", function (data) { return res.status(200).write(data); }); downloadStream.on("error", function (err) { return res.status(404).send({ message: "Cannot download the Image!" }); }); downloadStream.on("end", () => { return res.end(); }); } catch (error) { return res.status(500).send({ message: error.message, }); } }; module.exports = { uploadFiles, getListFiles, download, }; Define routes In routes folder, define routes in index.js with Express Router. const express = require("express"); const router = express.Router(); const homeController = require("../controllers/home"); const uploadController = require("../controllers/upload"); let routes = app => { router.get("/", homeController.getHome); router.post("/upload", uploadController.uploadFiles); router.get("/files", uploadController.getListFiles); router.get("/files/:name", uploadController.download); return app.use("/", router); }; module.exports = routes; There are 4 routes: – GET: Home page for the upload form. – POST "/upload" to call the uploadFiles function of the controller. This is also for action="/upload" in the view. – GET /files for list of Images. – GET /files/:name to download the image with the file name. Create Express app server Finally, we create an Express server. server.js const cors = require("cors"); const express = require("express"); const app = express(); const initRoutes = require("./routes"); var corsOptions = { origin: " " }; app.use(cors(corsOptions)); app.use(express.urlencoded({ extended: true })); initRoutes(app); let port = 8080; app.listen(port, () => { console.log(`Running at localhost:${port}`); }); What we do are: – import express and cors modules: Express is for building the Rest apis cors provides Express middleware to enable CORS with various options. – create an Express app, then add cors middlewares using app.use() method. Notice that we set origin: . – listen on port 8080 for incoming requests. Run & Check result On the project root folder, run this command: node src/server.js The console shows: Running at localhost:8080 Open browser with url . Click on Submit button, if the file is uploaded and stored in MongoDB successfully, the console shows image’s information: { fieldname: 'file', originalname: 'extreme.jpg', encoding: '7bit', mimetype: 'image/jpeg', id: 5dd60e6aee27a520ccad86a0, filename: '1574309482432-bezkoder-extreme.jpg', metadata: null, bucketName: 'photos', chunkSize: 261120, size: 89431, md5: '0d072efbb134b3b186c5da705e0e7059', uploadDate: 2019-11-21T04:11:23.264Z, contentType: 'image/jpeg' } Check MongoDB database, you will see bezkoder_files_db with 2 collections: photos.chunks & photo.files: Node.js upload/store multiple images in MongoDB Now I will show you how to modify some code to deal with multiple images instead of only one image at a time. Change form for uploading multiple images For the form, we write the input tag with new multiple attribute like this. views/index.html Modify middleware for uploading & storing image For multiple images, we use another function: array() instead of single(). middleware/upload.js ... // var uploadFile = multer({ storage: storage }).single("file"); var uploadFiles = multer({ storage: storage }).array("file", 10); var uploadFilesMiddleware = util.promisify(uploadFiles); module.exports = uploadFilesMiddleware; – array() function limits the number of files to upload each time, the first parameter is the name of – input tag (in html view:), the second parameter is the max number of files (10). Controller for uploading multiple images With the controller, we update the uploadFiles function. controllers/upload.js const uploadFiles = async (req, res) => { try { await upload(req, res); console.log(req.files); if (req.files.length There is a additional part in the catch() block. We check if the error.code is "LIMIT_UNEXPECTED_FILE" for showing user the message when he tries to upload more than 10 images/files. Don't forget to change the controller method on router: router.post("/upload", uploadController.uploadFiles); Run & Check result Run the command: node src/server.js And the console shows: Running at localhost:8080 Open your browser with url add some images like this: Click on Submit button. If these images are uploaded and stored in MongoDB successfully, you can see: The console shows these images' information: [{ fieldname: 'multi-files', originalname: 'circle.png', encoding: '7bit', mimetype: 'image/png', id: 5dd637361ae0143fdc572105, filename: '1574319926050-bezkoder-circle.png', metadata: null, bucketName: 'photos', chunkSize: 261120, size: 193593, md5: 'f6455d94e5ec26a1b6eb75b334220cbd', uploadDate: 2019-11-21T07:05:30.273Z, contentType: 'image/png' }, { fieldname: 'multi-files', originalname: 'JavaDuke.png', encoding: '7bit', mimetype: 'image/png', id: 5dd637361ae0143fdc572106, filename: '1574319926110-bezkoder-JavaDuke.png', metadata: null, bucketName: 'photos', chunkSize: 261120, size: 112767, md5: 'b7f0fa8ea1932850d99a64b35484484a', uploadDate: 2019-11-21T07:05:30.275Z, contentType: 'image/png' }, { fieldname: 'multi-files', originalname: 'lotus.jpg', encoding: '7bit', mimetype: 'image/jpeg', id: 5dd637361ae0143fdc572107, filename: '1574319926121-bezkoder-lotus.jpg', metadata: null, bucketName: 'photos', chunkSize: 261120, size: 375719, md5: '26dc2e512d7c021daaa7cb46215d4a5b', uploadDate: 2019-11-21T07:05:32.326Z, contentType: 'image/jpeg' }] Check MongoDB database: - photos.files collection: - chunks.files collection: If you try to upload more than 10 files at a time, you can see the error like this: Conclusion Today we’ve learned how to upload and store single/multiple images in MongoDB database using express, multer & multer-gridfs-stor