Here is a complete, step-by-step guide to deploying your MERN application on a LAN for 500+ users.

This guide uses a professional, 3-server architecture. This is the standard, scalable way to handle hundreds of users, as it prevents the bottlenecks and crashes you'd see if you ran everything on one machine.

Our Architecture

We will use three (3) servers. These can be physical machines or Virtual Machines (VMs) on your network.

1. 🌐 **Server A: The "Proxy Server" (e.g., `192.168.1.10`)**
   - **Role:** The only machine users access.
   - **Software: Nginx**.
   - **Jobs:** Serves your static React app (the `build` folder) and acts as a "traffic cop" (reverse proxy) to send API requests to your backend.
2. ⚙ **Server B: The "Application Server" (e.g., `192.168.1.11`)**
   - **Role:** Runs your backend API.
   - **Software: Node.js** & **PM2**.
   - **Jobs:** Runs your Express.js code in a scalable "cluster" (one process per CPU core) to handle many requests at once.
3. 🗄 **Server C: The "Database Server" (e.g., `192.168.1.12`)**
   - **Role:** Stores your data.
   - **Software: MongoDB**.
   - **Jobs:** Runs the MongoDB database securely and efficiently, isolated from users.

Step 1: 🗄 Configure the Database Server (Server C)

On `192.168.1.12`, we'll install, secure, and configure MongoDB.

1. **Install MongoDB:** Install the MongoDB Community Server on this machine.
2. **Edit Configuration:** Open your MongoDB config file.
   - **Linux:** `/etc/mongod.conf`
   - **Windows:** `C:\Program Files\MongoDB\Server\<version>\bin\mongod.cfg`
3. **Bind IP & Enable Security:** Find the `net:` and `security:` sections and change them. This is the **most critical step**.

   YAML

   ```yaml
   # mongod.conf / mongod.cfg

   net:
     port: 27017
     # This allows connections from the server itself (127.0.0.1)
     # AND from your Application Server (192.168.1.11).
   ```

```
  bindIp: 127.0.0.1, 192.168.1.11

security:
  # This forces your app to log in with a username/password.
  authorization: enabled
```

4. **Restart MongoDB:**
   - **Linux (systemd):** `sudo systemctl restart mongod`
   - **Windows (Services):** Find "MongoDB Server" in `services.msc` and restart it.
5. **Create Your App User:** You must now create a user, or your app won't be able to connect.
   - Open the `mongo` shell (or `mongosh`) on Server C.
   - Run these commands, replacing `myAppUser`, `myStrongPassword`, and `myAppDb` with your own values:

   JavaScript

   ```
   // 1. Connect to the 'admin' database to create a user
   use admin

   // 2. Create the user
   db.createUser({
     user: "myAppUser",
     pwd: "myStrongPassword",
     roles: [
       { role: "readWrite", db: "myAppDb" }
     ]
   })

   // 3. Exit the shell
   exit
   ```

Server C is now done. It will only accept authenticated connections from Server B.

Step 2: ⚙ Deploy the Backend API (Server B)

On `192.168.1.11`, we'll run the Node.js API using a process manager called PM2.

1. **Install Node.js:** Install the latest LTS version of Node.js.
2. **Get Your Code:** Clone or copy your backend project folder to this server.
3. **Install Dependencies:** `cd /path/to/your-backend` and run `npm install`.
4. **Remove Static Serving:** In your `server.js` (or `app.js`), **delete** these lines. Nginx (Server A) will handle this now.

   JavaScript

   ```
   // DELETE THESE LINES from your server.js
   // app.use(express.static(path.join(__dirname, 'client/build')));
   // app.get('*', (req, res) => {
   //   res.sendFile(path.join(__dirname, 'client/build',
   'index.html'));
   ```

```
// });
```

5. **Set Environment Variables:** Create a `.env` file in your project's root. This is where you connect to Server C.

   Code snippet

   ```
   PORT=5000
   # Use the Server C IP and the user/password you just created
   MONGODB_URI="mongodb://myAppUser:myStrongPassword@192.168.1.12:27017/
   myAppDb"

   # Add any other secrets (JWT_SECRET, etc.)
   JWT_SECRET="YOUR_SUPER_SECRET_KEY"
   ```

6. **Install PM2:** This is the process manager that will run your app as a cluster. `npm install -g pm2`
7. **Start Your Cluster:** This is the key command for handling 500+ users. `pm2 start server.js -i max`
   - `pm2 start server.js`: Tells PM2 to run your app.
   - `-i max`: This is the "magic." It detects all CPU cores on Server B and starts one app process for each core. PM2 automatically load-balances requests between them.
8. **Save the Process:** (Optional, but recommended) `pm2 save` This saves your running app list so PM2 will auto-start it if the server reboots.

Server B is now done. It's running a high-performance, multi-process API cluster.

Step 3: 🌐 Deploy the Frontend & Proxy (Server A)

On `192.168.1.10`, we'll set up Nginx to serve the React app and act as the "front door."

1. **Install Nginx:** Install Nginx on this server.
2. **Build Your React App:** On your *development* machine:
   - **IMPORTANT:** Make sure all your API calls in your React code are relative (e.g., `axios.get('/api/users')`, not `axios.get('http://localhost:5000/api/users')`).
   - Run `npm run build`. This creates a `build` folder.
3. **Copy Files to Server:** Copy the *contents* of your `build` folder to Server A's web root.
   - **Linux:** `/var/www/html/` (you may need to clear the default files first)
   - **Windows:** `C:\nginx\html\`
4. **Configure Nginx:** Edit the Nginx configuration file.
   - **Linux:** `/etc/nginx/sites-available/default`
   - **Windows:** `C:\nginx\conf\nginx.conf`

   Replace the *entire* `server { ... }` block with this:

   Nginx

```
# nginx.conf

server {
    listen 80;
    # This is the IP for your entire LAN to access the app
    server_name 192.168.1.10;

    # --- Part 1: Serve the React Frontend ---
    # Path to your 'build' folder's contents
    root /var/www/html;
    index index.html;

    location / {
        # This is the "magic" for React Router.
        # It tries to find the file, then a folder, and if it fails,
        # it falls back to index.html so React can handle the route.
        try_files $uri $uri/ /index.html;
    }

    # --- Part 2: Reverse Proxy for the API ---
    # This block catches any request starting with /api/
    # and forwards it to your Node.js server (Server B).
    location /api/ {
        # This is the IP and Port of Server B (your backend)
        proxy_pass http://192.168.1.11:5000;

        # These headers are important for passing info correctly
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

5. **Test and Restart Nginx:**
    - **Linux:**
        1. `sudo nginx -t` (This tests your config file for errors)
        2. `sudo systemctl restart nginx`
    - **Windows:**
        1. `cd C:\nginx`
        2. `nginx.exe -t`
        3. `nginx.exe -s reload`

Step 4: 🛡 Firewall Configuration (Final Check)

For a true production setup, you should lock down your server firewalls.

- 🌐 **Server A (Nginx):** Allow incoming TCP connections on **port 80** from *everyone* on the LAN.
- ⚙ **Server B (Node.js):** Allow incoming TCP connections on **port 5000** *only* from Server A's IP (`192.168.1.10`). Block everyone else.

- 🗄 **Server C (Mongo):** Allow incoming TCP connections on **port 27017** *only* from Server B's IP (`192.168.1.11`). Block everyone else.

You Are Done!

Your deployment is complete.

To access the application, tell your 500+ users to open a web browser and go to:

`http://192.168.1.10`