DevOps & Node.js Deployment Documentation

This document contains the complete set of documentation for setting up MongoDB, Node.js (via NVM), and deploying applications on Windows and Ubuntu EC2 instances.

Table of Contents

MongoDB Installation on Windows

We will install two (2) things: MongoDB Server and MongoDB Shell (to use from the command line).

💻 Step 1: Download MongoDB Community Server

Open Browser: Type (or click) this link in your web browser:

https://www.mongodb.com/try/download/community

Select Options: On the download page, select these options:

- **Version**: Latest version (usually default).
- **OS (Operating System)**: Windows.
- **Package**: MSI (This is the installation file).

Download: Click the Download button and let the file save.

⚙️ Step 2: Install MongoDB Server

Run Installer: Double-click the downloaded file (.msi).

Setup Wizard: The Setup Wizard will start.

Setup Type:

- On the "Choose Setup Type" screen, select the "Complete" option.

Service Configuration (Important):

- Keep "Install MongoD as a Service" checked.
- Select "Run service as Network Service user" (default).
- Keep "Service Name" as MongoDB.

MongoDB Compass (Optional):

- Keep "Install MongoDB Compass" (GUI tool) checked (This helps in viewing files/data graphically).

Finish: Click Install and let the installation complete.

🛠️ Step 3: Install MongoDB Shell (mongosh)

MongoDB Shell `mongosh` is necessary for running commands.

Download Shell: Open this link in your browser:

https://www.mongodb.com/try/download/shell

Select Options:

- **OS**: Windows
- **Package**: MSI

Run Installer: Double-click the downloaded shell installer file and install with default settings.

🧪 Step 4: Verification (Check)

Now we will check if everything is working correctly or not.

Open Command Prompt: Go to Windows Search and type `cmd` to open Command Prompt (or PowerShell).

Run Shell: Type this command:

```
mongosh
```

If everything is correct, you will enter the MongoDB shell, and you will see something like this:

```
Current Mongosh version: X.X.X
Connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
...
>
```

Exit: To exit the shell, type:

```
exit
```

Now your MongoDB Server and Shell are ready on Windows 11!

What's the Next Step?

Do you want to see the code to connect MongoDB in Node.js, or continue with the To-Do List project?

MongoDB Installation on EC2 Ubuntu (user1, step by step)

Okay — first we will install MongoDB on EC2, then integrate it into the app. Slow, stepwise, without skipping.

---

1. System update

- **Update packages:**

```
sudo apt update && sudo apt upgrade -y
```

---

2. MongoDB install

- **Install MongoDB server (Ubuntu repo):**

```
sudo apt install -y mongodb
```

You might get an error here:

```
ubuntu@ip-172-31-38-139:~$ sudo apt install -y mongodb
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Package mongodb is not available, but is referred to by another package.
This may mean that the package is missing, has been obsoleted, or
is only available from another source

E: Package 'mongodb' has no installation candidate
ubuntu@ip-172-31-38-139:~$
```

Solution:

(Note: The original text had "solution:" followed by blank lines. Preserving as is.)

- **Start and enable service:**

```
sudo systemctl start mongodb
sudo systemctl enable mongodb
```

- **Status check:**

```
sudo systemctl status mongodb
```

  - Expected: “active (running)”

---

3. Local security check

- **Default bind IP:** MongoDB binds to 127.0.0.1 by default (local machine only). This is safe; we are not opening remote access right now.
- **Port:** 27017 opens locally. No need to open this in EC2 security group (app is on the same server).

---

4. Basic test (mongo shell)

- **Connect to local MongoDB:**

  ```
  mongosh
  ```

  If `mongosh` is not found:

  ```
  sudo apt install -y mongodb-clients
  mongosh
  ```

- **Quick DB test:**

  ```
  use shortener
  db.urls.insertOne({ originalUrl: "https://example.com", shortId: "ex1" })
  db.urls.find()
  ```

- **Exit:**

  ```
  exit
  ```

---

5. Database and user for App (optional but good practice)

If you want to create an app user:

- **Admin login:**

  ```
  mongosh
  ```

- **Create app DB and user:**

  ```
  use shortener
  db.createUser({
    user: "appuser",
    pwd: "Strong_Password_ChangeKaro",
    roles: [{ role: "readWrite", db: "shortener" }]
  })
  exit
  ```

- **Connection string (local):**

```
mongodb://appuser:Strong_Password_ChangeKaro@127.0.0.1:27017/shortener
```

---

6. Node.js app integration (minimal wiring now, code later)

- **Mongoose install (in app folder):**

```
npm install mongoose
```

- **Env variable set (temporary, shell session):**

```
export MONGO_URL="mongodb://127.0.0.1:27017/shortener"
```

Or if you created a user:

```
export
MONGO_URL="mongodb://appuser:Strong_Password_ChangeKaro@127.0.0.1:27017/shortener"
```

- **server.js connection (example):**

```
const mongoose = require('mongoose');
const mongoUrl = process.env.MONGO_URL || 'mongodb://127.0.0.1:27017/shortener';

mongoose.connect(mongoUrl, { useNewUrlParser: true, useUnifiedTopology: true })
  .then(() => console.log('MongoDB connected'))
  .catch(err => console.error('MongoDB error:', err));
```

---

7. Service hardening (later, once base is working)

- **Auth enable + config file:**
  - File: `/etc/mongodb.conf` or `/etc/mongod.conf` (Ubuntu variant)
  - Ensure:
    - **bindIp:** `127.0.0.1`
    - **authorization:** enabled
  - Restart:

```
sudo systemctl restart mongodb
```

- We will keep it local-only for now; absolutely no remote exposure.

---

8. Verify end-to-end

- **MongoDB status:** `sudo systemctl status mongodb`
- **Shell test:** `mongosh` → `use shortener` → `db.stats()`
- **App start:** PM2/Forever whichever you are using:

```
pm2 restart nodeapp
```

Logs check:

```
pm2 logs nodeapp
```

Expected: “MongoDB connected”

---

9. Common issues quick-fix

- **mongosh not found:** `sudo apt install -y mongodb-clients`
- **Service inactive:** `sudo systemctl start mongodb`
- **Connection refused:** Check `bindIp` is `127.0.0.1`, and app connecting to `127.0.0.1:27017`
- **Auth errors:** If user enabled, use correct username/password in connection string.

Brother, once this base setup is done, drop a message: “Mongo connected” logs are appearing or if there is any error. Then we will shift schema + routes to MongoDB step-by-step (from in-memory to persistent storage).

Node NVM Version Management

Great job boss 👌 — right decision. If you want different Node.js versions for multiple users, **removing the system-wide install is best practice**.

🛠 Stepwise Removal (Ubuntu EC2)

1. Check current Node.js install

```
node -v
npm -v
```

This will confirm which version is currently installed system-wide.

2. Remove Node.js and npm

```
sudo apt remove -y nodejs npm
sudo apt autoremove -y
```

This will uninstall both packages.

3. Verify removal

```
node -v
npm -v
```

Expected: `command not found`.

🎯 Next Step After Removal

Now each user (user1, user2) will **install NVM** in their own environment and use their required Node.js version.

- User1 → `nvm install 18 && nvm use 18`
- User2 → `nvm install 20 && nvm use 20`

This way both will remain independent with their separate versions, and there will be no conflict.

👉 Boss, do you want me to create an **NVM installation and setup guide** for user1 step-by-step, then for user2?

Perfect boss 👌 — now you have created **user4** and **user5**, and both are ready for your Node.js + DevOps experiments. This is exactly the right step because now you can test **different environments** for multiple users.

🛠 Next Logical Step

Now we will do this:

1. System-wide Node.js has been removed

- Meaning there is no Node.js under root/Ubuntu.
- Each user will manage their own Node.js version.

2. NVM install per user

We need to setup NVM separately for each user:

```
# Login as user4
ssh -i your-key.pem user4@<EC2-PUBLIC-IP>

# NVM install
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
source ~/.nvm/nvm.sh

# Node.js version install (example v18)
nvm install 18
nvm use 18
```

```
# Login as user5
ssh -i your-key.pem user5@<EC2-PUBLIC-IP>

# NVM install
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
source ~/.nvm/nvm.sh

# Node.js version install (example v20)
nvm install 20
nvm use 20
```

---

3. Verify per user isolation

- `node -v` under user4 → v18
- `node -v` under user5 → v20 👉 Both will use their own versions, without conflict.

---

4. Experiment Ideas

- Run same app with different Node versions (user4 vs user5).
- Test PM2/Forever under each user.
- Compare logs, memory usage, and compatibility.
- Document differences for your README.

🎯 Now you have a **multi-user, multi-version Node.js lab** ready on EC2. This will give you a chance to simulate real production scenarios.

👉 Boss, do you want me to create a **PM2 setup guide for user4 and user5** right now so both can run apps in the background with their separate Node.js versions?

NVM Version Management of Node

Awesome boss 👌 — now you want to clone your **URL shortener app** (which you made earlier) under **EC2's new users (user4 / user5)** and test it. Let's make a stepwise guide:

🛠 Step 1: Clone App

You need to clone separately under each user:

```
# Login as user4
ssh -i your-key.pem user4@<EC2-PUBLIC-IP>

# Clone App (from GitHub)
git clone https://github.com/<your-repo>/url-shortener.git

cd url-shortener
```

Repeat the same steps for user5.

🛠 Step 2: Install Dependencies

Inside the app folder:

```
npm install
```

This will install all dependencies according to your `package.json`.

🛠 Step 3: Set Environment Variables

If you want to use MongoDB, set the connection string:

```
export MONGO_URL="mongodb://127.0.0.1:27017/shortener"
```

Or if you want to use Atlas, paste the Atlas connection string.

🛠 Step 4: Run App

Run with PM2 (recommended):

```
pm2 start server.js --name shortener
```

Check logs:

```
pm2 logs shortener
```

Expected output:

```
MongoDB connected
Server running on port 4000
```

🛠 Step 5: Browser Test

- Get EC2 **public IP**.
- If Nginx reverse proxy is set → `http://<EC2-PUBLIC-IP>/`
- If direct run → `http://<EC2-PUBLIC-IP>:4000` (or whatever port you set).

🎯 Result

- Your **URL shortener app** will now run independently under user4 / user5.
- Both can test with their own Node.js versions.
- Data will persist with MongoDB, links will remain saved even after reboot.

👉 Boss, do you want me to make an **Nginx multi-user config** now so that user4 and user5 apps are served on different paths (`/user4` , `/user5`) on the EC2 public IP?

Troubleshooting and Advanced Setup

1. User Sudo Permission Error

Error:

```
user1@ip-172-31-38-139:~$ sudo apt update && sudo apt install -y nodejs npm
[sudo] password for user1:
user1 is not in the sudoers file.
user1@ip-172-31-38-139:~$
```

Solution: Switch to ubuntu login and add user to sudo:

```
sudo usermod -aG sudo user1
```

---

2. Repo Analysis (links-shorten-app-ec2-deployment)

Quick Take: Your app repo is clean and structured 👌. It is a **Node.js + Express based URL shortener**, with Tailwind CSS UI. If you use user1 on EC2, deployment will be simple — just install Node.js, clone repo, install dependencies, and run/start service.

- **Tech Stack**:
  - Backend: Node.js, Express.js
  - Frontend: HTML5, Vanilla JS, Tailwind CSS
  - Styling: Dark theme + animations
  - Tools: npm, Git
- **Features**:
  - 🔗 Instant URL shortening (unique short IDs)
  - ⚡ Fast redirection
  - 🎨 Premium dark-mode UI
  - 📋 One-click copy for links
  - 🛡️ Error handling (404 pages)
  - 💾 In-memory storage (can extend to DB later)
- **Repo Files**:
  - `server.js` → main entry point
  - `views/` → frontend templates
  - `public/` → static assets (CSS/JS)
  - `test.js` → testing logic
  - `package.json` → dependencies + scripts
  - `README.md` → setup instructions

---

3. Deployment Plan (EC2, user1)

Since you decided to use **user1**, the steps will be:

1. Login EC2 as user1

```
ssh -i your-key.pem user1@<EC2-PUBLIC-IP>
```

2. Install Node.js & npm

```
sudo apt update && sudo apt install -y nodejs npm
node -v
npm -v
```

3. Clone Repo

```
git clone https://github.com/KahafSameer/links-shorten-app-ec2-deployment.git
cd links-shorten-app-ec2-deployment
```

4. Install Dependencies

```
npm install
```

5. Run App

```
node server.js
```

Browser: `http://<EC2-PUBLIC-IP>:3000`

6. Background Service (Optional)

```
npm install -g pm2
pm2 start server.js
pm2 status
```

---

4. Next Steps

- **Security**: If using user1, ensure their permissions and SSH keys are secure.
- **Persistence**: Use pm2 or systemd so the app runs even after reboot.
- **Scaling**: Adding DB (MongoDB/Postgres) in the future will be easy, as currently it is in-memory storage.

👉 Kahaf, if you want, I can tell you **pm2 + Nginx reverse proxy setup** too so the app becomes production-grade. Would you like to do a simple run test now or learn direct production-style setup?

---

5. Nginx Reverse Proxy Setup (Parallel Run)

Absolutely right thinking Kahaf 👍 — you want the **app currently running to not be disturbed**, and run the **same app in a new directory with Nginx**. This approach is safe, because you will do a kind of "duplicate deployment":

one test instance (which is running now) and one production-style instance (with Nginx).

🛠 Stepwise Plan

1. Create New Directory

On EC2 under user1:

```
cd ~
mkdir nodeapp-nginx
cd nodeapp-nginx
```

2. Clone Repo (fresh copy)

```
git clone https://github.com/KahafSameer/links-shorten-app-ec2-deployment.git .
npm install
```

3. Run App (Change Port)

So that existing app (port 3000) is not disturbed, run this on a **different port** (e.g. 4000):

Change port in `server.js` :

```
const PORT = 4000;
```

Run test:

```
node server.js
```

Browser: `http://<EC2-PUBLIC-IP>:4000`

4. Nginx Reverse Proxy Setup

Create Nginx config file:

```
sudo nano /etc/nginx/sites-available/nodeapp-nginx
```

Config:

```
server {
    listen 80;
    server_name _;

    location / {
        proxy_pass http://127.0.0.1:4000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
```

```
    }
}
```

Enable config:

```
sudo ln -s /etc/nginx/sites-available/nodeapp-nginx /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl reload nginx
```

✅ Result

- Your **first app** is still running on port 3000 (direct browser access).
- Your **new app** is accessible via Nginx on port 80:

  ```
  http://<EC2-PUBLIC-IP>/
  ```

---

6. PM2 vs Forever

Absolutely 👍 — let me explain in easy wording so it becomes clear to you:

🔑 What are PM2 and Forever?

🟢 PM2

- This is a **process manager** for Node.js apps.
- Meaning: you won't need to run `node server.js` manually.
- It runs the app **in the background**, and if the server reboots, the app **automatically restarts**.
- Extra features:
  - View logs (`pm2 logs`)
  - Manage multiple apps (`pm2 list`)
  - Auto-restart if it crashes
  - Easy monitoring Think of PM2 as a **smart manager** that keeps your Node.js app alive always.

🔵 Forever

- This is also a **simple process manager** for Node.js.
- As per name: runs the app "forever".
- If app crashes, it restarts it.
- It is a bit lightweight, gives fewer features compared to PM2.
- Mainly used when you just want to run an app in background without monitoring.

⚖️ Difference (Easy Words)

Tool	Use Case	Features
PM2	Production apps	Auto-restart, monitoring, logs, multiple apps
Forever	Simple background run	Auto-restart, lightweight, fewer features

👉 Easy analogy:

- **Forever** = a simple watchman who just keeps the app running.

- **PM2** = a full manager who runs the app, monitors it, and handles it upon crashing.

---

7. PM2 Setup Guide

Perfect Kahaf 👌 — let's run your app with **PM2** now. This will give you production-style stability: app will run in background, auto-restart on crash, and survive after reboot.

1. Install PM2 (global)

```
npm install -g pm2
```

Check version:

```
pm2 -v
```

2. Start App with PM2

Go to app directory (which is to be run with Nginx):

```
cd ~/nodeapp-nginx
pm2 start server.js --name nodeapp
```

👉 `--name nodeapp` gives you a friendly name instead of process ID.

3. View Status and Logs

```
pm2 list
pm2 logs nodeapp
```

4. Manage App

- Stop: `pm2 stop nodeapp`
- Restart: `pm2 restart nodeapp`
- Delete: `pm2 delete nodeapp`

5. Auto-start on Reboot

```
pm2 startup systemd
```

This command will output a line (like `sudo env PATH=$PATH:/usr/bin pm2 startup systemd -u user1 --hp /home/user1`). Copy-paste and run that line.

Then save current processes:

```
pm2 save
```

✅ Result

- Your app will now **run in background**.
- If it crashes → PM2 will auto-restart.
- If EC2 reboots → App will auto-start via PM2 startup script.
- Nginx will now always serve on port 80, without running `node server.js` manually.

---

8. Forever Setup Guide

Awesome Kahaf 👌 — now you want to test your **second folder app** on port **3000** with **Forever**. Let me give a stepwise easy guide:

1. Install Forever (under user1)

```
npm install -g forever
```

Check version:

```
forever --version
```

2. Go to App Directory

The second folder you created (port 3000 one):

```
cd ~/nodeapp   # or whatever the folder name is
```

3. Start App with Forever

```
forever start server.js
```

👉 This will run your app in background on port 3000. Now even if you close the terminal, the app will keep running.

4. Check Status

```
forever list
```

This will show how many apps are running with Forever.

5. View Logs

```
forever logs
```

Or for specific app:

```
forever logs 0
```

(0 = process ID shown in `forever list`)

6. Manage

- Stop: `forever stop server.js`
- Restart: `forever restart server.js`
- Delete (remove from list): `forever stopall`

🎯 Result

- Your **second app** is now running in background on port 3000 with Forever.
- You now have **two setups parallel**:
  - Port 4000 → With PM2 (Nginx reverse proxy)
  - Port 3000 → With Forever (direct run)

👉 Kahaf, now you can do a **real comparison** of both tools:

- PM2 → monitoring, auto-restart on reboot, logs, multiple apps manage.
- Forever → simple background run, lightweight, basic restart.