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

Mango DB Installation on Windows

Hum do (2) cheezein install karenge: MongoDB Server aur MongoDB Shell (command line se use karne ke liye).

💻 Step 1: MongoDB Community Server Download Karna

Browser Kholo: Apne web browser mein yeh link type karo (ya click karo):

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

Options Select Karo: Download page par, yeh options select karo:

- **Version**: Latest version (usually default hota hai).
- **OS (Operating System)**: Windows.
- **Package**: MSI (Yah installation file hai).

Download: Download button par click karo aur file ko save hone do.

⚙️ Step 2: MongoDB Server Install Karna

Installer Chalao: Jo file (.msi) download hui hai, us par double-click karo.

Setup Wizard: Setup Wizard shuru hoga.

Setup Type:

- "Choose Setup Type" screen par, "Complete" option select karo.

Service Configuration (Zaroori):

- "Install MongoD as a Service" option ko checked rehne do.
- "Run service as Network Service user" (default) ko select karo.
- "Service Name" ko MongoDB hi rehne do.

MongoDB Compass (Optional):

- "Install MongoDB Compass" (GUI tool) ko checked rehne do (Yah files/data ko graphical tareeke se dekhne mein madad karta hai).

Finish: Install par click karo aur installation poora hone do.

🛠️ Step 3: MongoDB Shell (mongosh) Install Karna

MongoDB Shell mongosh zaroori hai commands chalaane ke liye.

Download Shell: Browser mein yeh link open karo:

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

Options Select Karo:

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

Installer Chalao: Jo shell installer file download hui hai, us par double-click karo aur default settings ke saath install kar lo.

🧪 Step 4: Verification (Check Karna)

Ab hum check karenge ki sab kuch theek se kaam kar raha hai ya nahi.

Command Prompt Kholo: Windows Search mein jao aur cmd type karke Command Prompt (ya PowerShell) kholo.

Shell Run Karo: Yeh command type karo:

```
mongosh
```

Agar sab kuch theek hai, to tum MongoDB shell ke andar chale jaaoge, aur tumhein kuch aisa dikhega:

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

Exit: Shell se bahar aane ke liye type karo:

```
exit
```

Ab tumhara MongoDB Server aur Shell Windows 11 par ready hai!

Next Step Kya Hoga?

Kya tum Node.js mein MongoDB connect karne ka code dekhna chahte ho, ya To-Do List project continue karna chahte ho?

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

Thik hai bhai — pehle EC2 pe MongoDB install karte hain, phir app me integrate karenge. Slow, stepwise, bina jump kiye.

---

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
```

Error aye ga yaha pr:

```
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 aur 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 by default 127.0.0.1 pe bind hota hai (local machine only). Ye safe hai; abhi hum remote access nahi khol rahe.
- **Port:** 27017 local pe open hota hai. EC2 security group me isko open karne ki zarurat nahi (app same server pe hai).

---

4. Basic test (mongo shell)

- **Connect to local MongoDB:**

  ```
  mongosh
  ```

  Agar `mongosh` na mile:

  ```
  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. App ke liye database aur user (optional but good practice)

Agar tum app user create karna chahte ho:

- **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 (app folder me):**

```
npm install mongoose
```

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

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

Ya agar user bana liya hai:

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

- **server.js me 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, jab base chal jaye)

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

```
sudo systemctl restart mongodb
```

- Abhi hum local-only rakhenge; remote exposure bilkul nahi.

---

8. Verify end-to-end

- **MongoDB status:** `sudo systemctl status mongodb`
- **Shell test:** `mongosh` → `use shortener` → `db.stats()`

- **App start:** PM2/Forever jo use kar rahe ho:

  ```
  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.

Bhai, ye base setup ho jaye to message drop karna: “Mongo connected” logs aa rahe hain ya koi error. Phir hum schema + routes ko MongoDB pe shift karenge step-by-step (in-memory se persistent storage).

Node NVM Version Management

Shabash boss 👌 — sahi faisla hai. Agar tum multiple users ke liye alag-alag Node.js versions chahte ho to **system-wide install remove karna best practice hai**.

🛠 Stepwise Removal (Ubuntu EC2)

1. Check current Node.js install

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

Ye confirm karega ke abhi kaunsa version system-wide install hai.

2. Remove Node.js and npm

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

Ye dono packages uninstall kar dega.

3. Verify removal

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

Expected: `command not found`.

🎯 Next Step After Removal

Ab har user (user1, user2) apne environment me **NVM install karega** aur apna required Node.js version use karega.

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

Is tarah dono apne alag versions ke sath independent rahenge, aur koi conflict nahi hoga.

👉 Boss, kya tum chahte ho main ab tumhe **NVM installation aur setup guide** bana ke du user1 ke liye step-by-step, phir user2 ke liye?

Perfect boss 👌 — ab tumne **user4** aur **user5** create kar liye hain, aur ye dono tumhare Node.js + DevOps experiments ke liye ready hain. Ye ekdum sahi step hai kyunki ab tum multiple users ke liye **different environments** test kar sakte ho.

🛠 Next Logical Step

Ab hum ye karenge:

1. System-wide Node.js remove ho chuka hai

- Matlab ab root/Ubuntu ke under Node.js nahi hai.
- Har user apna apna Node.js version manage karega.

2. NVM install per user

Har user ke liye alag NVM setup karna hoga:

```
# 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 👉 Dono apne apne versions use karenge, bina conflict ke.

---

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.

🎯 Ab tumhare paas ek **multi-user, multi-version Node.js lab** ready hai EC2 pe. Ye tumhe real production scenarios simulate karne ka mauka dega.

👉 Boss, kya tum chahte ho main tumhe abhi **user4 aur user5 ke liye PM2 setup guide** bana ke du taake dono apne alag Node.js versions ke sath background me apps run kar saken?

NVM Version Management of Node

Zabardast boss 👌 — ab tum apna **URL shortener app** jo pehle banaya tha, usko **EC2 ke naye users (user4 / user5)** ke under clone karke test karna chahte ho. Chalo stepwise guide banate hain:

🛠 Step 1: App Clone karna

Har user ke under alag-alag clone karna hoga:

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

# App clone karo (GitHub se)
git clone https://github.com/<tumhara-repo>/url-shortener.git

cd url-shortener
```

Same steps user5 ke liye bhi repeat karo.

🛠 Step 2: Dependencies Install

App folder ke andar:

```
npm install
```

Ye tumhare `package.json` ke hisaab se saari dependencies install karega.

🛠 Step 3: Environment Variables Set

Agar tum MongoDB use karna chahte ho to connection string set karo:

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

Ya agar Atlas use karna hai to Atlas ka connection string paste karo.

🛠 Step 4: App Run karna

PM2 ke sath run karo (recommended):

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

Logs check karo:

```
pm2 logs shortener
```

Expected output:

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

🛠 Step 5: Browser Test

- EC2 ka **public IP** lo.
- Agar Nginx reverse proxy set hai → `http://<EC2-PUBLIC-IP>/`
- Agar direct run hai → `http://<EC2-PUBLIC-IP>:4000` (ya jo port tumne set kiya hai).

🎯 Result

- Tumhara **URL shortener app** ab user4 / user5 ke under independently run karega.
- Dono apne apne Node.js versions ke sath test kar sakte hain.
- MongoDB ke sath data persist hoga, reboot ke baad bhi links save rahenge.

👉 Boss, kya tum chahte ho main tumhe ab **Nginx multi-user config** bana ke du taake user4 aur user5 ke apps alag paths (`/user4` , `/user5`) pe serve ho EC2 ke public IP par?

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: Tumhara app repo clean aur structured hai 👌. Ye ek **Node.js + Express based URL shortener** hai, Tailwind CSS UI ke sath. Agar tum user1 hi use karoge EC2 pe, toh deployment simple ho jaayega — bas Node.js install, repo clone, dependencies install, aur run/start service karna hoga.

- **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 tumne decide kiya hai **user1** hi use karna, steps ye rahenge:

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**: Agar user1 hi use karna hai, toh ensure karo ke uske permissions aur SSH keys secure hon.
- **Persistence**: pm2 ya systemd use karo taake app reboot ke baad bhi chale.
- **Scaling**: Future me DB (MongoDB/Postgres) add karna easy hoga, kyunki abhi in-memory storage hai.

👉 Kahaf, tum chaho to main tumhe **pm2 + Nginx reverse proxy setup** bhi bata dun taake app production-grade ho jaaye. Kya tum abhi simple run test karna pasand karoge ya direct production-style setup seekhna chahte ho?

---

5. Nginx Reverse Proxy Setup (Parallel Run)

Bilkul sahi soch hai Kahaf 👍 — tum chahte ho ke **jo app abhi chal raha hai wo disturb na ho**, aur ek **new directory me same app ko Nginx ke sath run karo**. Ye approach safe hai, kyunki tum ek tarah se “duplicate

deployment" karoge: ek test instance (jo abhi chal raha hai) aur ek production-style instance (Nginx ke sath).

🛠 Stepwise Plan

1. New Directory Banaana

EC2 pe user1 ke under:

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

2. Repo Clone Karna (fresh copy)

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

3. App Run Karna (Port Change)

Taake existing app (port 3000) disturb na ho, isko **different port** pe run karo (e.g. 4000):

`server.js` me port change:

```
const PORT = 4000;
```

Run test:

```
node server.js
```

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

4. Nginx Reverse Proxy Setup

Nginx config file banao:

```
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

- Tumhara **pehla app** abhi bhi port 3000 pe chal raha hai (direct browser access).
- Tumhara **new app** Nginx ke through port 80 pe accessible hai:

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

---

6. PM2 vs Forever

Bilkul 👍 — chalo easy wording me samjhata hoon taake tumhe clear ho jaaye:

🔑 PM2 aur Forever kya hote hain?

🟢 PM2

- Ye ek **process manager** hai Node.js apps ke liye.
- Matlab: tumhe `node server.js` manually chalane ki zarurat nahi hogi.
- Ye app ko **background me run** karata hai, aur agar server reboot ho jaaye to app **automatically restart** ho jaata hai.
- Extra features:
  - Logs dekhna (`pm2 logs`)
  - Multiple apps manage karna (`pm2 list`)
  - Auto-restart agar crash ho jaaye
  - Easy monitoring Soch lo PM2 ek **smart manager** hai jo tumhare Node.js app ko hamesha zinda rakhta hai.

🔵 Forever

- Ye bhi ek **simple process manager** hai Node.js ke liye.
- Naam ke hisaab se: app ko “forever” chalata hai.
- Agar app crash ho jaaye to restart kar deta hai.
- Thoda lightweight hai, PM2 ke comparison me kam features deta hai.
- Mainly use hota hai jab tumhe sirf ek app ko background me chalana ho bina monitoring ke.

⚖️ 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** = ek simple watchman jo sirf app ko chalata rakhta hai.
- **PM2** = ek full manager jo app ko chalata bhi hai, monitor bhi karta hai, aur crash hone pe handle bhi karta hai.

---

7. PM2 Setup Guide

Perfect Kahaf 👌 — chalo ab tumhare app ko **PM2** ke sath run karte hain. Ye tumhe production-style stability dega: app background me chalega, crash hone pe auto-restart hoga, aur reboot ke baad bhi survive karega.

1. PM2 Install karo (global)

```
npm install -g pm2
```

Check version:

```
pm2 -v
```

2. App ko PM2 ke sath start karo

App directory me jao (jo Nginx ke sath run karna hai):

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

👉 `--name nodeapp` se tumhe ek friendly name milta hai instead of process ID.

3. Status aur Logs dekhna

```
pm2 list
pm2 logs nodeapp
```

4. App ko manage karna

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

5. Auto-start on Reboot

```
pm2 startup systemd
```

Ye command tumhe ek line output karega (jaise `sudo env PATH=$PATH:/usr/bin pm2 startup systemd -u user1 --hp /home/user1`). Us line ko copy-paste karke run karo.

Phir save current processes:

```
pm2 save
```

✅ Result

- Tumhara app ab **background me run** karega.
- Agar crash ho jaaye → PM2 auto-restart karega.
- Agar EC2 reboot ho jaaye → PM2 startup script se app auto-start ho jaayega.
- Nginx ab hamesha port 80 pe serve karega, bina manual `node server.js` chalaye.

---

8. Forever Setup Guide

Zabardast Kahaf 👌 — ab tum apne **dusre folder wale app** ko port **3000** par **Forever** ke sath test karna chahte ho. Chalo stepwise easy guide deta hoon:

1. Forever Install karo (user1 ke under)

```
npm install -g forever
```

Check version:

```
forever --version
```

2. App Directory me jao

Tumne jo dusra folder banaya tha (port 3000 wala):

```
cd ~/nodeapp   # ya jo bhi folder ka naam hai
```

3. Forever ke sath App Start karo

```
forever start server.js
```

👉 Ye tumhara app background me run karega port 3000 par. Ab tum terminal band bhi kar do, app chalti rahegi.

4. Status Check karo

```
forever list
```

Ye tumhe dikhayega kitne apps Forever ke sath chal rahe hain.

5. Logs dekhna

```
forever logs
```

Ya specific app ke liye:

```
forever logs 0
```

(0 = process ID jo `forever list` me dikhega)

6. Manage karna

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

🎯 Result

- Tumhara **dusra app** ab port 3000 par Forever ke sath background me chal raha hoga.
- Tumhare paas ab **do setups parallel** hain:
  - Port 4000 → PM2 ke sath (Nginx reverse proxy)
  - Port 3000 → Forever ke sath (direct run)

👉 Kahaf, ab tum dono tools ka **real comparison** kar sakte ho:

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