Multi-User Node.js Lab on Ubuntu EC2 with PM2 and Forever

Overview

This lab demonstrates setting up a multi-user, multi-version Node.js environment on Ubuntu EC2. It uses NVM for per-user Node.js isolation, PM2 and Forever for process management, and runs parallel apps on different ports. Multi-user isolation prevents version conflicts and simulates production scenarios.

> Why multi-user isolation matters: It allows testing different Node.js versions and apps independently, mimicking real-world deployments where users or teams manage separate services.

Prerequisites

- Ubuntu EC2 instance (e.g., t2.micro or t3.micro).
- SSH access with sudo privileges.
- Git installed (`sudo apt update && sudo apt install -y git`).

Remove System-Wide Node.js

Remove any globally installed Node.js to avoid conflicts with per-user NVM installations.

```
sudo apt remove -y nodejs npm
sudo apt autoremove -y
node -v || echo "Node removed"
npm -v || echo "NPM removed"
```

> Why this step: Global Node.js can interfere with NVM-managed versions, leading to unexpected behavior.

Create Users and SSH Setup

Create two users (user4 and user5) for isolated environments. Set passwords and configure SSH keys for secure access. Optionally grant minimal sudo for package installations.

```
sudo adduser user4
sudo adduser user5
# Set passwords interactively or via passwd
# Copy SSH keys to ~/.ssh/authorized_keys for each user
# Optional: sudo usermod -aG sudo user4 (for minimal sudo)
```

Per-User NVM Setup

Install NVM for each user to manage Node.js versions independently. Install v18 for user4 and v20 for user5.

```
# As user4
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
source ~/.nvm/nvm.sh
nvm install 18
nvm use 18
node -v

# As user5
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
source ~/.nvm/nvm.sh
nvm install 20
nvm use 20
node -v
```

> Why this step: NVM enables version isolation, allowing different users to run apps on compatible Node.js versions without conflicts.

Clone and Run Apps

Clone a demo Node.js app (e.g., a simple Express server) for each user. Run App A with PM2 on port 4000 under user4, and App B with Forever on port 3000 under user5.

```
# As user4 (for PM2 app)
git clone https://github.com/<your-org>/<your-url-shortener-or-demo>.git appA
cd appA
npm install
npm install -g pm2
pm2 start server.js --name nodeapp -- --port 4000

# As user5 (for Forever app)
git clone https://github.com/<your-org>/<your-url-shortener-or-demo>.git appB
cd appB
npm install
npm install -g forever
forever start server.js --port 3000
```

Logs and Monitoring

Monitor running processes and view logs for debugging.

```
# PM2 (as user4)
pm2 list
pm2 logs nodeapp
pm2 monit  # Optional interactive monitor

# Forever (as user5)
forever list
forever logs
```

PM2 Startup and Persistence

Configure PM2 for automatic startup and persistence across reboots.

```
# As user4
pm2 startup
pm2 save
```

Why this step: Ensures apps restart on system boot, improving reliability in production-like setups.

Comparison: PM2 vs Forever

Tool	Monitoring	Startup Persistence	Crash Recovery	Configuration	Best For
PM2	Yes	Yes (pm2 startup/save)	Strong	Ecosystem config	Production/staging
Forever	Minimal	No native (use systemd)	Basic	Simple CLI	Quick tests

Troubleshooting

- **NVM not loaded:** Ensure `source ~/.nvm/nvm.sh` in `.bashrc` or use login shells for PM2/Forever.
- **PM2 uses wrong Node:** Run `pm2 start` from a shell with NVM active; add NVM sourcing to `.bashrc` for services.
- **Ports blocked:** Open ports in EC2 security groups or use Nginx reverse proxy to port 80.
- **Permissions:** Run apps as their respective users; avoid sudo to prevent ownership issues.
- **Restart persistence:** Execute `pm2 startup` and `pm2 save` after configuration changes.

Next Steps

- Integrate MongoDB for data persistence.
- Set up Nginx reverse proxy for unified access (e.g., /app1 to port 4000, /app2 to port 3000).
- Add SSL with Let's Encrypt.
- Simulate Blue-Green deployments using user isolation.

What We Learned

This lab highlights the benefits of per-user Node.js management with NVM, process managers like PM2 and Forever for reliability, and multi-user setups for isolated testing. It prepares DevOps beginners for scalable, conflict-free deployments on EC2.