

Nginx

Reverse Proxy Server for Node.js

Node.js + Express: Deployment Challenge

We run web app with

```
node server.js
```

- Express listens on a port (e.g., 3000)
- Directly serves:
 - Static files (images, JS, CSS)
 - API requests

The Problem with This Setup

Issue	Description
Single Point of Failure	If the Node process crashes, your whole app is down.
Performance	Node is not efficient at serving static files (like images, CSS).
Security	Exposing Node directly to the internet increases attack surface.
Port Conflict	Usually runs on non-standard ports (e.g., 3000 instead of 80/443).
Scalability	Hard to handle multiple instances or load balancing.

Nginx as the Solution

Nginx acts as a **reverse proxy** in front of your Node.js app.

Reverse Proxy Concept

```
Client (Browser)
  ↓
[ Nginx :80 or 8080 ]
  ↓
[ Node.js :3000 ]
```

Nginx receives all requests from the outside world.

- It forwards API requests to Node.js (upstream).
- It can also directly serve static files from disk.

Benefits of Using Nginx

Feature	What It Does
Reverse Proxy	Routes traffic safely to Node.js running internally.
Static File Server	Serves HTML, CSS, images faster than Node.
SSL Termination	Handles HTTPS certificates easily.
Load Balancing	Distributes requests to multiple Node instances.
Process Isolation	Keeps Node.js behind a secure layer.

Example: Reverse Proxy Config

nginx.conf

We need to replace the default `nginx.conf` file in the default nginx configuration directory with this file: make a backup copy of `nginx.conf` to revert back.

```
worker_processes 1;
error_log logs/error.log;

events {
    worker_connections 1024;
}

http {
    include mime.types;
    default_type application/octet-stream;
    sendfile on;
    keepalive_timeout 65;
    include servers/*; # <---- or sites-enabled/*;
}
```

Default configuration directory

- macOS (Homebrew):

`/opt/homebrew/etc/nginx/nginx.conf`

- Linux (Ubuntu/Debian): `/etc/nginx/nginx.conf`

servers (or sites-enabled) subdirectory

The http block has the include command to add all the configuration files in the servers (or sites-enabled) directory.

```
http {  
    ...  
    include servers/*; # <---- or sites-enabled/*;  
}
```

nodejs-app.conf Overview

We create `nodejs-app.conf` that is included in the `nginx.conf`.

```
upstream nodejs_app {
    server localhost:3000;
    keepalive 8;
}

server {
    listen 8080;                # Port 8080 (works without sudo)
    server_name localhost;

    location / {
        proxy_pass http://nodejs_app;
        proxy_http_version 1.1;

        # Pass original client information
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

1. Upstream Block

```
upstream nodejs_app {  
    server localhost:3000;  
    keepalive 8;  
}
```

- Defines a **backend group** that Nginx can forward requests to.
- Here, it points to **localhost:3000**, where your Node.js app runs.
- `keepalive 8` keeps up to 8 idle connections open for reuse.

Think of this `nodejs_app` as a "shortcut name" for your backend app.

2. Server Block

```
server {  
    listen 8080;  
    server_name localhost;
```

- Listens on **port 8080**, no `sudo` needed (unlike port 80).
- Handles requests sent to `http://localhost:8080`.
- Each request goes to the correct **location** block below.

3. Location Block

```
location / {  
    proxy_pass http://nodejs_app;  
    proxy_http_version 1.1;  
}
```

- Forwards all paths (/ , /api , etc.) to your Node.js backend.
- Uses HTTP/1.1 to support **keep-alive** and **WebSocket** connections.

Flow:

```
Client → Nginx (8080) → Node.js (3000)
```

4. Forwarding Client Info

```
proxy_set_header Host $host;  
proxy_set_header X-Real-IP $remote_addr;  
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

- Preserves important information from the original client:
 - **Host** → Original domain name
 - **X-Real-IP** → Real client IP (not Nginx)
 - **X-Forwarded-For** → Chain of proxy IPs for logging & analytics

Node.js sees the **real client IP** even though Nginx sits in front.

Copy conf file and Restart Nginx

Copy the Nginx file

We should copy your Nginx configuration file (e.g., `nodejs-app.conf`) into the **servers** or **sites-enabled** directory, it's automatically loaded by Nginx — no extra edits needed.

Mac

We can setup the environment variable to find the configuration directory.

```
CONFIG_DIRECTORY=/opt/homebrew/etc/nginx # mac  
CONFIG_DIRECTORY=/usr/local/etc/nginx # mac/Intel
```

Ubuntu

Linux (Ubuntu) uses symbolic link to include the configuration file by convention.

- Copy the nodejs-app.conf file in the `sites-available` directory.
- Link the file in the `sites-enabled` file to include the configuration file from the `nginx.conf`.
- We can set the `CONFIG_DIRECTORY` to `sites-enabled` directory.

```
sudo mkdir -p /etc/nginx/sites-available /etc/nginx/sites-enabled
sudo cp ./nodejs-app.conf /etc/nginx/sites-available/nodejs-app.conf
sudo ln -sf /etc/nginx/sites-available/nodejs-app.conf /etc/nginx/sites-enabled/nodejs-app.conf
```

Validate syntax

The \$CONFIG_DIRECTORY is default nginx configuration directory.

```
> nginx -t -c $CONFIG_DIRECTORY/nginx.conf  
nginx: the configuration file /opt/homebrew/etc/nginx/nginx.conf syntax is ok  
nginx: configuration file /opt/homebrew/etc/nginx/nginx.conf test is successful
```

Run / Reload Nginx

macOS (Homebrew services)

Start/stop/restart nginx as a service

```
# Start (if not running)
brew services start nginx
# Stop
# Reload after changes
brew services restart nginx
```

Manual reload without services

```
nginx -s reload
```

Ubuntu/Debian (systemd)

```
sudo systemctl enable nginx --now  
sudo systemctl reload nginx    # after config changes  
# or  
sudo systemctl restart nginx
```

Start your Node.js app (port 3000)

The Node.js application

code/development/app/index.js

```
// hello.js
const http = require('http');

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello\n');
});

server.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

Start and test the node server app

```
node app.js
```

Test:

```
curl -i http://localhost:8080/
```

(Optional) Switch to port 80

- Change `listen 8080;` → `listen 80;` in your server block.
- **macOS (Homebrew):** privileged ports (<1024) require root.
Options:

- Run Nginx as root for reloads:

```
sudo /opt/homebrew/bin/nginx -t -c /opt/homebrew/etc/nginx/nginx.conf  
sudo /opt/homebrew/bin/nginx -s reload
```

- Or keep using **8080** and map from 80 at your router / local dev proxy.
- **Linux:** normal with `sudo systemctl reload nginx`.

Keep 8080 for development to avoid root privileges.

Docker

However, managing Nginx manually can be tedious and even error-prone.

- In real-world deployments, we often use Docker to manage Nginx automatically — handling configuration, restarts, and environment consistency.