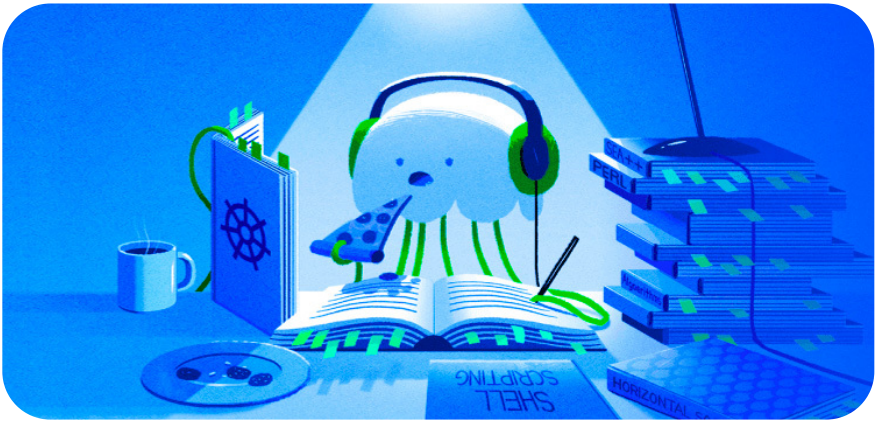
CONTENTS

Published on April 26, 2022

Ubuntu Ubuntu 22.04 Applications Nginx Node.js

By Alex Garnett
Senior DevOps Technical Writer



Node.js is an open-source JavaScript runtime environment for building server-side and networking applications. The platform runs on Linux, macOS, FreeBSD, and Windows. Though you can run Node.js applications at the command line, this tutorial will focus on running them as a service. This means that they will restart on reboot or failure and are safe for use in a production environment.

In this tutorial, you will set up a production-ready Node.js environment on a single Ubuntu 22.04 server. This server will run a Node.js application managed by PM2, and provide users with secure access to the application through an Nginx reverse proxy. The Nginx server will offer HTTPS using a free certificate provided by Let's Encrypt.

Prerequisites

This guide assumes that you have the following:

- Nginx configured with SSL using Let's Encrypt certificates. How To Secure Nginx with Let's Encrypt on Ubuntu 22.04 will walk you through the process.
- Node.js installed on your server. How To Install Node.js on Ubuntu 22.04

When you've completed the prerequisites, you will have a server serving your domain's default placeholder page at `https://example.com/`.

Step 1 – Creating a Node.js Application

Let's write a *Hello World* application that returns "Hello World" to any HTTP requests. This sample application will help you get up and running with Node.js. You can replace it with your own application — just make sure that you modify your application to listen on the appropriate IP addresses and ports.

First, using nano or your favorite text editor, create a

```
const http = require('http');

const hostname = 'localhost';
const port = 3000;

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

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${p
});
```

Copy

Save the file and exit the editor. If you are using `nano`, press `Ctrl+X`, then when prompted, `Y` and then Enter.

This Node.js application listens on the specified address (`localhost`) and port (`3000`), and returns “Hello World!” with a `200` HTTP success code. Since we’re listening on `localhost`, remote clients won’t be able to connect to our

```
Server running at http://localhost:3000/
```

Note: Running a Node.js application in this manner will block additional commands until the application is killed by pressing `CTRL+C`.

To test the application, open another terminal session on your server, and connect to `localhost` with `curl`:

```
$ curl http://localhost:3000
```
Copy

If you get the following output, the application is working properly and listening on the correct address and port:

```
Output
Hello World!
```

If you do not get the expected output, make sure that your Node.js application is running and configured to

applications so that they will run in the background as a service.

Use `npm` to install the latest version of PM2 on your


```
$ pm2 start hello.js
```

Copy

This also adds your application to PM2's process list, which is outputted every time you start an application:

```
Output
```

As indicated above, PM2 automatically assigns an `App name` (based on the filename, without the `.js` extension) and a PM2 `id`. PM2 also maintains other information, such as the `PID` of the process, its current status, and memory usage.

Applications that are running under PM2 will be restarted automatically if the application crashes or is killed, but we can take an additional step to get the application to launch on system startup using the `startup` subcommand. This subcommand generates and configures a startup script to launch PM2 and its managed processes on server boots:

```
$ pm2 startup systemd
```
Copy

The last line of the resulting output will include a command to run with superuser privileges in order to set PM2 to start on boot:

Output

```
$ sudo env PATH=$PATH:/usr/bin /usr/lib/node_m
```
Copy 'p

As an additional step, we can save the PM2 process list and corresponding environments:

```
$ pm2 save
```
Copy

You have now created a systemd *unit* that runs `pm2` for your user on boot. This `pm2` instance, in turn, runs `hello.js`.

Start the service with `systemctl`:

```
$ sudo systemctl start pm2-sammy
```
Copy

Check the status of the systemd unit:

```
$ systemctl status pm2-sammy
```
Copy

Stop an application with this command (specify the PM2 `App name` or `id`):

```
$ pm2 stop app_name_or_id
```

Copy

Restart an application:

```
$ pm2 restart app_name_or_id
```

Copy

List the applications currently managed by PM2:

```
$ pm2 list
```

Copy

Get information about a specific application using its `App name`:

```
$ pm2 info app_name
```

Copy

The PM2 process monitor can be pulled up with the

Now that your Node.js application is running and managed by PM2, let's set up the reverse proxy.

Step 3 – Setting Up Nginx as a Reverse Proxy Server

Your application is running and listening on `localhost`, but you need to set up a way for your users to access it. We will set up the Nginx web server as a reverse proxy for this purpose.

In the prerequisite tutorial, you set up your Nginx configuration in the `/etc/nginx/sites-available/example.com` file. Open this file for editing:

```
$ sudo nano /etc/nginx/sites-available/example
```
Copy

Within the `server` block, you should have an existing `location /` block. Replace the contents of that block with the following configuration. If your application is set to

```
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
...
}
```

This configures the server to respond to requests at its root. Assuming our server is available at `example.com`, accessing `https://example.com/` via a web browser would send the request to `hello.js`, listening on port `3000` at `localhost`.

You can add additional `location` blocks to the same server block to provide access to other applications on the same server. For example, if you were also running another Node.js application on port `3001`, you could add this location block to allow access to it via `https://example.com/app2`:

/etc/nginx/sites-available/example.com — Optional

```
...
}
```

Once you are done adding the location blocks for your applications, save the file and exit your editor.

Make sure you didn't introduce any syntax errors by typing:

```
$ sudo nginx -t
```
Copy

Restart Nginx:

```
$ sudo systemctl restart nginx
```
Copy

Assuming that your Node.js application is running, and your application and Nginx configurations are correct, you should now be able to access your application via the Nginx reverse proxy. Try it out by accessing your server's URL (its public IP address or domain name).

Next, you may want to look into How to build a Node.js application with Docker.

About the authors

Ask a question

Search for more help

Was this helpful? Yes No

Comments

4 Comments

You can type `!ref` in this text area to quickly search our full set of tutorials, documentation & marketplace offerings and insert the link!

Sign In or Sign Up to Comment

koesys • February 15, 2023

Hello, I've follow this tutorial, thanks, but I have an issue I try to resolve for 48h without success : I use react, so I changed res.end by the res.sendFile("index.html") of express (I did the express.static and all stuff) When I go to the website url, I have the index.html page but... every url is 404 (app.js, app.css...)

I am desperate. Please, can you help me ?

Reply

> connect() failed (111: Unknown error) while connecting to upstream... no live upstreams while connecting to upstream...

Closest thing I've found to an answer online is here, but it still doesn't seem to work.

Reply

Beresansky Anton • September 14, 2022

I would also add this to proxy config to pass-on visitors real-IP:

```
proxy_set_header        X-Real-IP       $remot
proxy_set_header        X-Forwarded-For $proxy
```

Thanks for the amazing tutorials. These are super helpful and guide in the right direction. I would suggest mentioning that user might have to restart VM before running `sudo systemctl start pm2-sammy` command. I faced a similar issue as mentioned in https://github.com/Unitech/pm2/issues/3924 and restarting resolved the problem.

Reply

Sign up

Popular Topics

Ubuntu

Linux Basics

JavaScript

Python

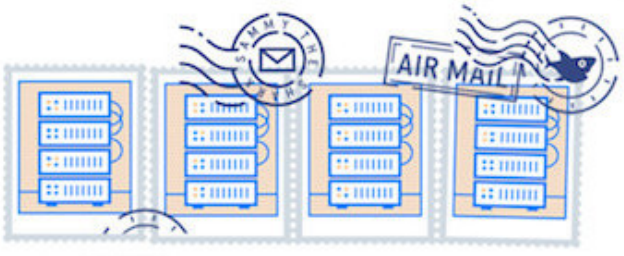
MySQL

Docker

Kubernetes

All tutorials →

Free Managed Hosting →



Get our biweekly newsletter

Sign up for Infrastructure as a Newsletter.

Sign up →

Working on improving health and education, reducing inequality, and spurring economic growth? We'd like to help.

Learn more →

Become a contributor

You get paid; we donate to tech nonprofits.

Kubernetes Course Learn Python 3
Machine Learning in Python Getting started with Go
Intro to Kubernetes

DigitalOcean Products

Cloudways Virtual Machines Managed Databases
Managed Kubernetes Block Storage Object Storage
Marketplace VPC Load Balancers

Welcome to the developer cloud

DigitalOcean makes it simple to launch in the cloud and scale up as you grow – whether you're running one virtual machine or ten thousand.

Release Notes

Newsletter Signup

Uptime

Meetups