

Mitchell Anicas
Dec 3, 2014

♥ 104 💬 42

Share



How To Set Up a Node.js Application for Production on Ubuntu 14.04

Tags: Node.js, Nginx

Distribution: Ubuntu

Introduction

Node.js is an open source Javascript runtime environment for easily building server-side and networking applications. The platform runs on Linux, OS X, FreeBSD, and Windows, and its applications are written in JavaScript. Node.js applications can be run at the command line but we will teach you how to run them as a service, so they will automatically restart on reboot or failure, so you can use them in a production environment.

In this tutorial, we will cover setting up a production-ready Node.js environment that is composed of two Ubuntu 14.04 servers; one server will run Node.js applications managed by PM2, while the other will provide users with access to the application through an Nginx reverse proxy to the application server.

Prerequisites

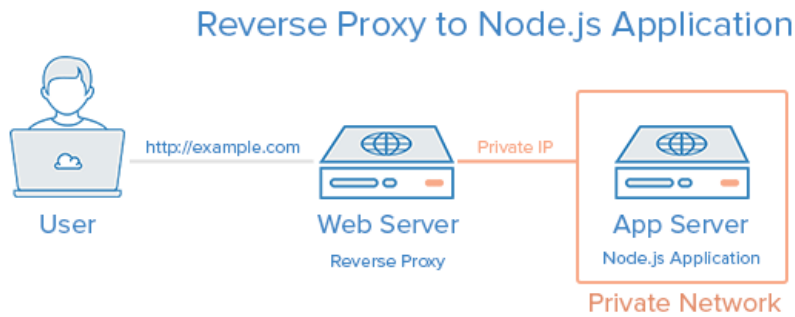
This guide uses two servers **with private networking** (in the same datacenter). We will refer to them by the following names:

- **app**: The server where we will install Node.js runtime, your Node.js application, and PM2
- **web**: The server where we will install the Nginx web server, which will act as a reverse proxy to your application. Users will access this server's public IP address to get to your Node.js application.

It is possible to use a single server for this tutorial, but you will have to make a few changes along the way. Simply use the localhost IP

address, i.e. `127.0.0.1`, wherever the **app** server's private IP address is used.

Here is a diagram of what your setup will be after following this tutorial:



Before you begin this guide, you should have a regular, non-root user with `sudo` privileges configured on both of your servers--this is the user that you should log in to your servers as. You can learn how to configure a regular user account by following steps 1-4 in our [initial server setup guide for Ubuntu 14.04](#).

If you want to be able to access your **web** server via a domain name, instead of its public IP address, purchase a domain name then follow these tutorials:

- [How To Set Up a Host Name with DigitalOcean](#)
- [How to Point to DigitalOcean Nameservers From Common Domain Registrars](#)

Let's get started by installing the Node.js runtime on the **app** server.

Install Node.js

We will build and install the latest stable release of Node.js, on the **app** server.

On the **app** server, let's update the apt-get package lists with this command:

```
sudo apt-get update
```

Then use `apt-get` to install Node.js build dependencies:

```
sudo apt-get install -y build-essential openssl libssl-dev pkg-config
```

The next step is download the source code for the latest release of Node.js.

Go to the [Node.js Downloads](#) page and find the **Source Code** download link. Right-click it, and copy its link address to your clipboard.

Change to your home directory and download the Node.js source with `wget`. Paste the download link in place of the highlighted part:

```
cd ~  
wget http://nodejs.org/dist/latest/node-v0.10.33.tar.gz
```

Now extract the tar archive you just downloaded with this command:

```
tar xvf node-v*
```

Change to the new directory that was created by extracting the archive:

```
cd node-v*
```

Configure and build Node.js with the following commands:

```
./configure  
make
```

Now, to install Node.js, run this command:

```
sudo make install
```

If you want to delete the Node.js source code that you downloaded, change to your home directory and use this `rm` command:

```
cd ~  
rm -rf node-v*
```

The Node.js runtime is now installed, and ready to run an application! Let's write a Node.js application.

Create Node.js Application

Now we will create a *Hello World* application that simply returns "Hello World" to any HTTP requests. This is a sample application that will help you get your Node.js set up, which 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.

Because we want our Node.js application to serve requests that come from our reverse proxy server, **web**, we will utilize our **app** server's private network interface for inter-server communication. Look up your **app** server's private network address.

If you are using a DigitalOcean droplet as your server, you may look up the server's private IP address through the *Metadata* service. On the **app** server, use the `curl` command to retrieve the IP address now:

```
curl -w "\n" http://169.254.169.254/metadata/v1/interfaces/private/0/ipv4/address
```

You will want to copy the output (the private IP address), as it will be used to configure our Node.js application.

Hello World Code

Next, create and open your Node.js application for editing. For this tutorial, we will use `vi` to edit a sample application called `hello.js`:

```
vi hello.js
```

Insert the following code into the file, and be sure to substitute the **app** server's private IP address for both of highlighted

`APP_PRIVATE_IP_ADDRESS` items. If you want to, you may also replace the highlighted port, `8080`, in both locations (be sure to use a non-admin port, i.e. 1024 or greater):

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(8080, 'APP_PRIVATE_IP_ADDRESS');
console.log('Server running at http://APP_PRIVATE_IP_ADDRESS:8080/');
```

Now save and exit.

This Node.js application simply listens on the specified IP address and port, and returns "Hello World" with a `200` HTTP success code. This means that the application is only reachable from servers on the same private network, such as our **web** server.

Test Application (Optional)

If you want to test if your application works, run this `node` command on the **app** server:

```
node hello.js
```

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

In order to test the application, open another terminal session and connect to your **web** server. Because the web server is on the same private network, it should be able to reach the private IP address of the **app** server using `curl`. Be sure to substitute in the **app** server's private IP address for `APP_PRIVATE_IP_ADDRESS`, and the port if you changed it:

```
curl http://APP_PRIVATE_IP_ADDRESS:8080
```

If you see the following output, the application is working properly and listening on the proper IP address and port:

```
Hello World
```

If you do not see the proper output, make sure that your Node.js application is running, and configured to listen on the proper IP address and port.

On the **app** server, be sure to kill the application (if you haven't already) by pressing `CTRL+C`.

Install PM2

Now we will install PM2, which is a process manager for Node.js applications. PM2 provides an easy way to manage and daemonize applications (run them as a service).

We will use Node Packaged Modules (NPM), which is basically a package manager for Node modules that installs with Node.js, to install PM2 on our **app** server. Use this command to install PM2:

```
sudo npm install pm2 -g
```

Manage Application with PM2

PM2 is simple and easy to use. We will cover a few basic uses of PM2.

Start Application

The first thing you will want to do is use the `pm2 start` command to run your application, `hello.js`, in the background:

```
pm2 start hello.js
```

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

App name	id	mode	PID	status	restarted	uptime	memory	watching
hello	0	fork	5871	online	0	0s	9.012 MB	disabled

As you can see, 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 an additional step needs to be taken to get the application to launch on system startup (boot or reboot). Luckily, PM2 provides an easy way to do this, the `startup` subcommand.

The `startup` subcommand generates and configures a startup script to launch PM2 and its managed processes on server boots. You must also specify the platform you are running on, which is `ubuntu`, in our case:

```
pm2 startup ubuntu
```

The last line of the resulting output will include a command (that must be run with superuser privileges) that you must run:

```
[PM2] You have to run this command as root
[PM2] Execute the following command :
[PM2] sudo env PATH=$PATH:/usr/local/bin pm2 startup ubuntu -u sammy
```

Run the command that was generated (similar to the highlighted output above) to set PM2 up to start on boot (use the command from your own output):

```
sudo env PATH=$PATH:/usr/local/bin pm2 startup ubuntu -u sammy
```

Other PM2 Usage (Optional)

PM2 provides many subcommands that allow you to manage or look up information about your applications. Note that running `pm2` without any arguments will display a help page, including example usage, that covers PM2 usage in more detail than this section of the tutorial.

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

```
pm2 stop example
```

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

```
pm2 restart example
```

The list of applications currently managed by PM2 can also be looked up with the `list` subcommand:

```
pm2 list
```

More information about a specific application can be found by using the `info` subcommand (specify the PM2 *App name* or *id*):

```
pm2 info example
```

The PM2 process monitor can be pulled up with the `monit` subcommand. This displays the application status, CPU, and memory usage:

```
pm2 monit
```

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

Set Up Reverse Proxy Server

Now that your application is running, and listening on a private IP address, you need to set up a way for your users to access it. We will set up an Nginx web server as a reverse proxy for this purpose. This tutorial will set up an Nginx server from scratch. If you already have an Nginx server setup, you can just copy the `location` block into the server block of your choice (make sure the location does not conflict with any of your web server's existing content).

On the **web** server, let's update the apt-get package lists with this command:

```
sudo apt-get update
```

Then install Nginx using apt-get:

```
sudo apt-get install nginx
```

Now open the default server block configuration file for editing:

```
sudo vi /etc/nginx/sites-available/default
```

Delete everything in the file and insert the following configuration. Be sure to substitute your own domain name for the `server_name` directive (or IP address if you don't have a domain set up), and the **app** server private IP address for the `APP_PRIVATE_IP_ADDRESS`. Additionally, change the port (`8080`) if your application is set to listen on a different port:

```
server {
    listen 80;

    server_name example.com;

    location / {
        proxy_pass http://APP_PRIVATE_IP_ADDRESS:8080;
        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;
    }
}
```

This configures the **web** server to respond to requests at its root. Assuming our server is available at `example.com`, accessing `http://example.com/` via a web browser would send the request to the application server's private IP address on port `8080`, which would be received and replied to by the Node.js application.

You can add additional `location` blocks to the same server block to provide access to other applications on the same **web** server. For example, if you were also running another Node.js application on the **app** server on port `8081`, you could add this location block to allow access to it via `http://example.com/app2`:

```
location /app2 {
    proxy_pass http://APP_PRIVATE_IP_ADDRESS:8081;
    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;
}
```

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

On the **web** server, restart Nginx:

```
sudo service nginx restart
```

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

Conclusion

Congratulations! You now have your Node.js application running behind an Nginx reverse proxy. This reverse proxy setup is flexible enough to provide your users access to other applications or static web content that you want to share. Good luck with your Node.js development!

Also, if you are looking to encrypt transmissions between your web server and your users, [here is a tutorial that will help you get HTTPS \(TLS/SSL\) support set up.](#)

Author:
Mitchell Anicas

Spin up an SSD cloud server in under a minute.

Simple setup. Full root access.
Straightforward pricing.

DEPLOY SERVER

Related Tutorials

[How To Redirect www to Non-www with Nginx on CentOS 7](#)





[How To Redirect www to Non-www with Nginx on Ubuntu 14.04](#)

[Automating the Deployment of a Scalable WordPress Site](#)

[How To Deploy a Rails App with Puma and Nginx on Ubuntu 14.04](#)

[How To Deploy a Rails App with Unicorn and Nginx on Ubuntu 14.04](#)

42 Comments

B I      



Leave a comment...

Log In to Comment



manvir December 24, 2014

Why use a proxy?

♡ 2 Reply



alexbaumgertner December 27, 2014

It's insecure to run any node app at 80 port (because only root have access to ports 0-1024, and in this case nodejs must run with root privileges).

See <https://groups.google.com/forum/?fromgroups=#!topic/nodejs/gB8veFbcX5g>

♡ 1



rodrigopnq December 29, 2014

Is it insecure even with setcap to run on port 80?

♡



asad January 13, 2015

+1 for it being insecure. But if a process can be hijacked thru http putting nginx in front of it won't help, except the attacker won't have root access, but if it has vulnerabilities exposed over tcp or other underlying protocol, nginx can save you.

♡



tonydelun January 3, 2015

What could be the reason I'm getting the error EADDRNOTAVAIL when executing `node hello.js` ?

♡ Reply



error74 January 9, 2015

I'm having the same problem. It seem to work when I switch it to my public ip... Did you ever figure out how to fix it?

♡



manicas MOD January 9, 2015

Are both of your servers in the same data center with private networking?

♡



apolhill January 14, 2015

Same problem for me, On the initial droplet setup I forgot to set private networking, so I stopped the droplet turned private networking on and restarted, but still get the same issue, any ideas. This is before any interaction with the web server, so can only assume the app server is at fault somewhere.



apolhill January 14, 2015

Creating from scratch with private networking turned on, fixes the issue.



manicas **MOD** January 14, 2015

If you create a droplet without private networking then enable private networking later, you must set up the new network interface on your system: [How To Enable DigitalOcean Private Networking on Existing Droplets](#).



windmaomao March 17, 2015

god, can they just say something on the private network page, point to this article as manicas pointed out, took me forever to find out.



Scorpi January 12, 2015

If we are using nginx here, maybe we can use some additional cache settings?
So nginx can serve all static files, and node will wake up only for executing some logic.

Reply



asb **MOD** January 13, 2015

Definetly. This is just an example of how to set up the basic infrastructure. For a deeper dive into what you can do with Nginx, check out:

- [Understanding Nginx HTTP Proxying, Load Balancing, Buffering, and Caching](#)



ytzvan January 12, 2015

If I want to run my node.js app across 3 nodes with multi-core processors, PM2 can help me achieve that? I know PM2 helps me run my app across a single server with multi-core support, but if I have more nodes, Can I use PM2 or need a load-balancer like Haproxy or Nginx Proxy?

Reply



asad January 13, 2015

Digital Oceans always on top of their game. I have been using Monit and Upstart directly, but this seems like a quicker alternative. Thanks.

1 Reply



danielmd January 20, 2015

What are the advantages of suing PM2 vs using something like nodejs forever + nohup?

Reply



manicas **MOD** January 20, 2015

Basically, PM2 has more features. Both can be used to keep an application running.

2



tombradev March 4, 2015

Great tutorial, but why you are not using NPM to install node instead? [@manicas](#)
Secondly, if we are using only a single server to be APP and WEB, what would it affect?

Reply



manicas **MOD** March 4, 2015

You can use a single server to run both components (I think it's mentioned in the tutorial somewhere). Separating them makes scaling easier, and isolates the web tier from the applications.



tombradev March 4, 2015



Thanks @manicas

I found out that using node v0.12 requires twice of this steps to complete installation of node:

*Configure and build Node.js with the following commands:

```
./configure  
make
```

Now, to install Node.js, run this command
sudo make install*

I hope it helps for the future install.

Cheers

♡ 1



nathanhombby March 20, 2015

npm is *part* of node... so is it even possible to use it to *install* node?

♡



OJS March 7, 2015

hi

Please could you tell me ..

Is it Ok (good ?) to use for creating the **app server** the image "**Node -v 0.12 on 14.04**" provided under the **"Create Droplet - Application"** menu ?

♡ Reply



nathanhombby March 20, 2015

That's what I'm doing and it's working fine. As far as I'm aware it just means you get to skip the installing node bit.

♡



nathaniel.rich March 7, 2015

Does it also work with node+socket.io (two way communication)?

♡ Reply



contactame March 8, 2015

Hi! i have this:

```
g++: internal compiler error: Killed (program cc1plus)
```

Please submit a full bug report,

with preprocessed source if appropriate.

See <file:///usr/share/doc/gcc-4.8/README.Bugs> for instructions.

```
make[1]: *** [/root/node-v0.12.0/out/Release/obj.target/v8_base/deps/v8/src/api.o] Error 4
```

```
make[1]: Leaving directory `/root/node-v0.12.0/out'
```

```
make: *** [node] Error 2
```

What happened?

♡ Reply



Buju March 11, 2015

Wonderful stuff! Question is how can I prevent directory browsing of my root folder when running my app in Ubuntu? Can nginx help me save my world in this case? if I run my web server on port 8000 and my rest api on 8001 how can nginx help me start both in one command :(

♡ Reply



manicas MOD March 11, 2015

Just create a separate server block for each app (listening port). Check out this tutorial: [How To Set Up Nginx Server Blocks](#)

♡ 1 Reply



TimothyGu April 9, 2015

@manicas I'm kind of confused on what the following do to the HTTP headers:

```
proxy_set_header Upgrade $http_upgrade;
proxy_set_header Connection 'upgrade';
proxy_cache_bypass $http_upgrade;
```

What is HTTP Upgrade? How does that have to do with reverse proxy?

♥ Reply



manicas **MOD** April 9, 2015

Those headers are necessary when proxying a WebSocket.



TimothyGu April 9, 2015

Also how does the nginx built-in load balancer compare to pm2's?

♥ Reply



manicas **MOD** April 9, 2015

I haven't used it, but I believe the PM2 load balancer performs load balancing at the Node process level, where Nginx is used for network load balancing.



fredrikstolpe April 9, 2015

This was a really excellent guide, just the info I needed to get started, thanks!

♥ 1 Reply



bazjapan April 14, 2015

Thanks for the guide. I assuming that if I configure nginx accordingly I can use this set up for a) multiple domains, and b) multiple app servers. Am I correct in that assumption?

♥ Reply



manicas **MOD** April 14, 2015

Yes, that is correct.



makenova April 14, 2015

@manicas, great tutorial.

My connection was hanging until I commented out the following line,

```
proxy_set_header Connection 'upgrade';
```

Do you have any idea why this would happen?

♥ Reply



Bchay9 April 18, 2015

When using nginx I am able to access my nodejs app at the URL myapp.com/home, but if I navigate to myapp.com I still get the nginx welcome page. Is there a fix for this?

♥ Reply



manicas **MOD** April 20, 2015

Did you set your `location` directive to `"/home"` ? It should look like this:

```
location / {
```



vikramkohli April 25, 2015

Thanks Mitchell. Wonderful post and has helped us setting our Nodejs production environment. We are facing issue where Nginx is

not severing updated HTML files which is part of one nodejs application.

Do Nginx cache all static contents by default? If yes, then what is the location of caching in Ubuntu? Or it doesn't cache, can you pls help with providing required conf setting for this. Googled alot but not able to find the solution.

♥ Reply



kamaln7 **MOD** April 27, 2015

By default, nginx does not cache any files. You can configure nginx so serve static files and pass everything else to Node.js like this:

```
server {
    listen 80;

    server_name example.com;
    root /path/to/nodejs/app/static

    location / {
        try_files $uri @nodejs;
    }

    location @nodejs {
        proxy_pass http://APP_PRIVATE_IP_ADDRESS:8080;
        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;
    }
}
```

♥ 1



poloyc May 6, 2015

Great post! Now I'm truly ready for production. Digital Ocean you are really good!

♥ Reply

Load More Comments



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.



Copyright © 2015 DigitalOcean™ Inc.

[Community](#) [Tutorials](#) [Questions](#) [Projects](#) [Tags](#) [Terms, Privacy, & Copyright](#) [Security](#)

