CONTENTS

RELATED

How To Install nginx on CentOS 6 with yum
View

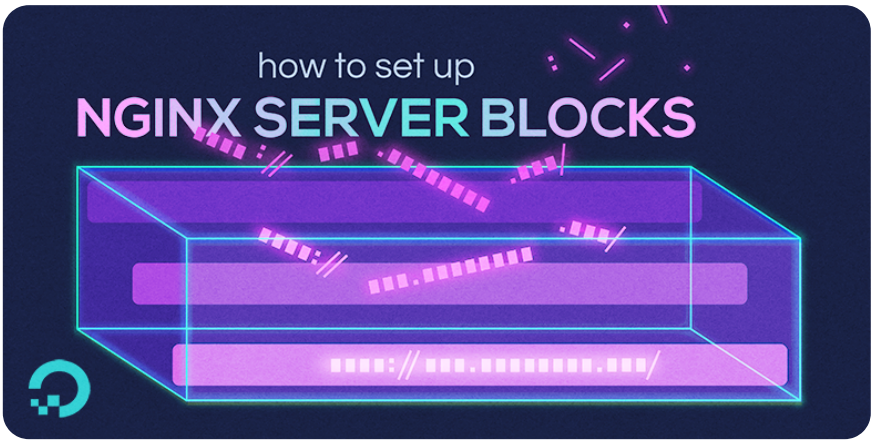
Initial Server Setup with Ubuntu 12.04
View

// Tutorial //

How To Set Up Nginx Server Blocks (Virtual Hosts) on Ubuntu 16.04

Published on May 19, 2016 · Updated on July 9, 2021

Ubuntu Getting Started Nginx Ubuntu 16.04



By Justin Ellingwood
Developer and author at DigitalOcean.

Not using Ubuntu 16.04?
Choose a different version or distribution.

Ubuntu 16.04

Introduction

When using the Nginx web server, **server blocks** (similar to virtual hosts in Apache) can be used to encapsulate configuration details and host more than one domain on a single server.

In this guide, we'll discuss how to configure server blocks in Nginx on an Ubuntu 16.04 server.

Prerequisites

We're going to be using a non-root user with `sudo` privileges throughout this tutorial. If you do not have a user like this configured, you can create one by following our Ubuntu 16.04 initial server setup guide.

You will also need to have Nginx installed on your server. The following guides cover this procedure:

- How To Install Nginx on Ubuntu 16.04: Use this guide to set up Nginx on its own.
- How To Install Linux, Nginx, MySQL, PHP (LEMP stack) in Ubuntu 16.04: Use this guide if you will be using Nginx in conjunction with MySQL and PHP.

When you have fulfilled these requirements, you can continue on with this guide.

Example Configuration

For demonstration purposes, we're going to set up two domains with our Nginx server. The domain names we'll use in this guide are **example.com** and **test.com**.

Note: for more information on setting up a domain with DigitalOcean, please see our Domains and DNS product documentation.

Popular Topics

Ubuntu
Linux Basics
JavaScript
React
Python
Security
MySQL
Docker
Kubernetes
Browse all topic tags

Free Managed Hosting →

All tutorials →

Questions

Q&A Forum
Ask a question
DigitalOcean Support

If you do not have two spare domain names to play with, use placeholder names for now and we'll show you later how to configure your local computer to test your configuration.

Step 1 – Setting Up New Document Root Directories

By default, Nginx on Ubuntu 16.04 has one server block enabled. It is configured to serve documents out of a directory at `/var/www/html`.

While this works well for a single site, we need additional directories if we're going to serve multiple sites. We can consider the `/var/www/html` directory the default directory that will be served if the client request doesn't match any of our other sites.

We will create a directory structure within `/var/www` for each of our sites. The actual web content will be placed in an `html` directory within these site-specific directories. This gives us some additional flexibility to create other directories associated with our sites as siblings to the `html` directory if necessary.

We need to create these directories for each of our sites. The `-p` flag tells `mkdir` to create any necessary parent directories along the way:

```
$ sudo mkdir -p /var/www/example.com/html
$ sudo mkdir -p /var/www/test.com/html
```

Copy

Now that we have our directories, we will reassign ownership of the web directories to our normal user account. This will let us write to them without `sudo`.

Note: Depending on your needs, you might need to adjust the permissions or ownership of the folders again to allow certain access to the `www-data` user. For instance, dynamic sites will often need this. The specific permissions and ownership requirements entirely depend on your configuration. Follow the recommendations for the specific technology you're using.

We can use the `$USER` environmental variable to assign ownership to the account that we are currently signed in on (make sure you're not logged in as **root**). This will allow us to easily create or edit the content in this directory:

```
$ sudo chown -R $USER:$USER /var/www/example.com/html
$ sudo chown -R $USER:$USER /var/www/test.com/html
```

Copy

The permissions of our web roots should be correct already if you have not modified your `umask` value, but we can make sure by typing:

```
$ sudo chmod -R 755 /var/www
```

Copy

Our directory structure is now configured and we can move on.

Step 2 – Creating Sample Pages for Each Site

Now that we have our directory structure set up, let's create a default page for each of our sites so that we will have something to display.

Create an `index.html` file in your first domain:

```
$ nano /var/www/example.com/html/index.html
```

Copy

Inside the file, we'll create a really basic file that indicates what site we are currently accessing. It will look like this:

/var/www/example.com/html/index.html

```
<html>
    <head>
        <title>Welcome to Example.com!</title>
    </head>
    <body>
        <h1>Success! The example.com server block is working!</h1>
    </body>
</html>
```

Save and close the file when you are finished. To do this in `nano`, press `CTRL+o` to write the file out, then `CTRL+x` to exit.

Since the file for our second site is basically going to be the same, we can copy it over to our second document root like this:

```
$ cp /var/www/example.com/html/index.html /var/www/test.com/html/
```

Copy

Now, we can open the new file in our editor:

```
$ nano /var/www/test.com/html/index.html
```
Copy

Modify it so that it refers to our second domain:

/var/www/test.com/html/index.html

```
<html>
    <head>
        <title>Welcome to Test.com!</title>
    </head>
    <body>
        <h1>Success!  The test.com server block is working!</h1>
    </body>
</html>
```

Save and close this file when you are finished. We now have some pages to display to visitors of our two domains.

Step 3 – Creating Server Block Files for Each Domain

Now that we have the content we wish to serve, we need to create the server blocks that will tell Nginx how to do this.

By default, Nginx contains one server block called `default` which we can use as a template for our own configurations. We will begin by designing our first domain's server block, which we will then copy over for our second domain and make the necessary modifications.

Creating the First Server Block File

As mentioned above, we will create our first server block config file by copying over the default file:

```
$ sudo cp /etc/nginx/sites-available/default /etc/nginx/sites-available/example.com
```
Copy

Now, open the new file you created in your text editor with `sudo` privileges:

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

Ignoring the commented lines, the file will look similar to this:

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

```
server {
        listen 80 default_server;
        listen [::]:80 default_server;

        root /var/www/html;
        index index.html index.htm index.nginx-debian.html;

        server_name _;

        location / {
                try_files $uri $uri/ =404;
        }
}
```

First, we need to look at the listen directives. **Only one of our server blocks on the server can have the `default_server` option enabled.** This specifies which block should serve a request if the `server_name` requested does not match any of the available server blocks. This shouldn't happen very frequently in real world scenarios since visitors will be accessing your site through your domain name.

You can choose to designate one of your sites as the "default" by including the `default_server` option in the `listen` directive, or you can leave the default server block enabled, which will serve the content of the `/var/www/html` directory if the requested host cannot be found.

In this guide, we'll leave the default server block in place to serve non-matching requests, so we'll remove the `default_server` from this and the next server block. You can choose to add the option to whichever of your server blocks makes sense to you.

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

```
server {
        listen 80;
        listen [::]:80;

        . . .
}
```

Note: You can check that the `default_server` option is only enabled in a single active file by typing:

```
$ grep -R default_server /etc/nginx/sites-enabled/
```
Copy

If matches are found uncommented in more than on file (shown in the leftmost column), Nginx will complain about an invalid configuration.

The next thing we're going to have to adjust is the document root, specified by the `root` directive. Point it to the site's document root that you created:

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

```
server {
        listen 80;
        listen [::]:80;

        root /var/www/example.com/html;

}
```

Next, we need to modify the `server_name` to match requests for our first domain. We can additionally add any aliases that we want to match. We will add a `www.example.com` alias to demonstrate.

When you are finished, your file will look something like this:

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

```
server {
        listen 80;
        listen [::]:80;

        root /var/www/example.com/html;
        index index.html index.htm index.nginx-debian.html;

        server_name example.com www.example.com;

        location / {
                try_files $uri $uri/ =404;
        }
}
```

That is all we need for a basic configuration. Save and close the file to exit.

Creating the Second Server Block File

Now that we have our initial server block configuration, we can use that as a basis for our second file. Copy it over to create a new file:

```
$ sudo cp /etc/nginx/sites-available/example.com /etc/nginx/sites-available/test.com
```
Copy

Open the new file with `sudo` privileges in your editor:

```
$ sudo nano /etc/nginx/sites-available/test.com
```
Copy

Again, make sure that you do not use the `default_server` option for the `listen` directive in this file if you've already used it elsewhere. Adjust the `root` directive to point to your second domain's document root and adjust the `server_name` to match your second site's domain name (make sure to include any aliases).

When you are finished, your file will likely look something like this:

/etc/nginx/sites-available/test.com

```
server {
        listen 80;
        listen [::]:80;

        root /var/www/test.com/html;
        index index.html index.htm index.nginx-debian.html;

        server_name test.com www.test.com;

        location / {
                try_files $uri $uri/ =404;
        }
}
```

When you are finished, save and close the file.

Step 4 – Enabling your Server Blocks and Restart Nginx

Now that we have our server block files, we need to enable them. We can do this by creating symbolic links from these files to the `sites-enabled` directory, which Nginx reads from during startup.

We can create these links by typing:

```
$ sudo ln -s /etc/nginx/sites-available/example.com /etc/nginx/sites-enabled/
$ sudo ln -s /etc/nginx/sites-available/test.com /etc/nginx/sites-enabled/
```

Copy

These files are now linked into the enabled directory. We now have three server blocks enabled, which are configured to respond based on their `listen` directive and the `server_name` (you can read more about how Nginx processes these directives here):

- `example.com`: Will respond to requests for `example.com` and `www.example.com`
- `test.com`: Will respond to requests for `test.com` and `www.test.com`
- `default`: Will respond to any requests on port 80 that do not match the other two blocks.

In order to avoid a possible hash bucket memory problem that can arise from adding additional server names, we will also adjust a single value within our `/etc/nginx/nginx.conf` file. Open the file now:

```
$ sudo nano /etc/nginx/nginx.conf
```

Copy

Within the file, find the `server_names_hash_bucket_size` directive. Remove the `#` symbol to uncomment the line:

/etc/nginx/nginx.conf

```
http {
    . . .

    server_names_hash_bucket_size 64;

    . . .
}
```

Save and close the file when you are finished.

Next, test to make sure that there are no syntax errors in any of your Nginx files:

```
$ sudo nginx -t
```

Copy

If no problems were found, restart Nginx to enable your changes:

```
$ sudo systemctl restart nginx
```

Copy

Nginx should now be serving both of your domain names.

Step 5 – Modifying Your Local Hosts File for Testing (Optional)

If you have not been using domain names that you own and instead have been using placeholder values, you can modify your local computer's configuration to let you to temporarily test your Nginx server block configuration.

This will not allow other visitors to view your site correctly, but it will give you the ability to reach each site independently and test your configuration. This works by intercepting requests that would usually go to DNS to resolve domain names. Instead, we can set the IP addresses we want our local computer to go to when we request the domain names.

Note: Make sure you are operating on your local computer during these steps and not a remote server. You will need to have root access, be a member of the administrative group, or otherwise be able to edit system files to do this.

If you are on a Mac or Linux computer at home, you can edit the file needed by typing:

```
local $ sudo nano /etc/hosts
```

Copy

If you are on Windows, you can find instructions for altering your hosts file here.

You need to know your server's public IP address and the domains you want to route to the server. Assuming that my server's public IP address is `203.0.113.5`, the lines I would add to my file would look something like this:

/etc/hosts

```
127.0.0.1   localhost
. . .
```

```
203.0.113.5 example.com www.example.com
203.0.113.5 test.com www.test.com
```

This will intercept any requests for `example.com` and `test.com` and send them to your server, which is what we want if we don't actually own the domains that we are using.

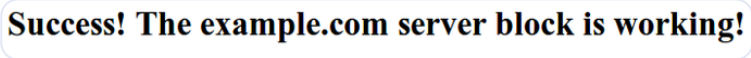
Save and close the file when you are finished.

Step 6 – Testing Your Results

Now that you are all set up, you should test that your server blocks are functioning correctly. You can do that by visiting the domains in your web browser:

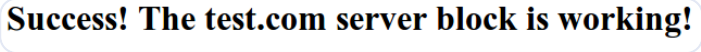
```
http://example.com
```

You should see a page that looks like this:



If you visit your second domain name, you should see a slightly different site:

```
http://test.com
```



If both of these sites work, you have successfully configured two independent server blocks with Nginx.

At this point, if you adjusted your `hosts` file on your local computer in order to test, you'll probably want to remove the lines you added.

If you need domain name access to your server for a public-facing site, you will probably want to purchase a domain name for each of your sites.

Conclusion

You should now have the ability to create server blocks for each domain you wish to host from the same server. There aren't any real limits on the number of server blocks you can create, so long as your hardware can handle the traffic.

About the authors

Justin Ellingwood Author

Developer and author at DigitalOcean.

Still looking for an answer? Ask a question Search for more help

Was this helpful? Yes No

Comments

10 Comments

Leave a comment...

This textbox defaults to using **Markdown** to format your answer.

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

san22 • July 3, 2016

Thanks ... very well written... I noticed a typo... to check that the default_server option is only enabled in a single active file by typing: grep -R default_server /etc/nginx/sites-enabled/

should be grep -R default_server /etc/nginx/sites-available/

Show replies Reply

meteoroxide • December 20, 2016

Its totally shocking. In every article since 2012, you always skip the part where you are supposed to create the virtual host and instead copy the default.

Reply

phlozzle • October 22, 2016

Thanks for this, it's super helpful. I wanted to host a Django server and node server on one Ubuntu instance, and this got me started in the right direction. I wrote a little bit about setting up multiple server blocks

Reply

georgetour • August 9, 2016

This how guides should be made.Thank you.

Reply

fuadquazi • August 24, 2017

Nginx is serving the html files, and so when I put in a php script it just serves the file and it gets downloaded. How do I make nginx run the php files ?

bambangyudhotomo • March 13, 2017

this tutorial work great. But www.example.com not working. Only example.com work. I already added CNAME for www could you help me please?

Show replies Reply

luis02lopez • January 28, 2017

@jellingwood Thanks in advance for the tutorial. I'm experiment some problems, I follow all the steps and this is my server configuration:

```
http://pastebin.com/6E3mdjrR
```

Even though when I go to:

```
http://kinbu.localhost/
```

I got "HTTP ERROR 500" page It's not working.

Plase help. Thanks.

Reply

block2trade • January 21, 2017

can anyone help at the end I get an error 404 not found? it happens to any site even if I use http://ipadd/test.php same error

Reply

brandonb3daf1f9 • October 25, 2016

I am using LEMP on Ubutu 16.04 with a virtual host(server block). When I log into phpmyadmin, I get a 404 page. But when I go back to /phpmyadmin I am logged in. So it is logging me in, but redirecting to a 404. The url (after logging in) is index.php(Plus a ton of other characters). Any idea what is going on?

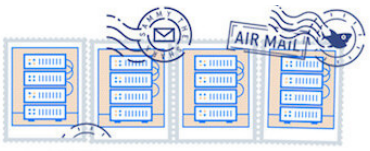
Reply

govindsaini1986 • September 21, 2016

Hi I have followed all the steps but site is not running. when I access domain yournexthosting.com in browser it download a file automatically. when I access yournexthosting.com/index.html it downloads index.html page... plz help.

Show replies Reply

Load More Comments



GET OUR BIWEEKLY NEWSLETTER

Sign up for Infrastructure as a Newsletter.

HOLLIE'S HUB FOR GOOD

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

BECOME A CONTRIBUTOR

You get paid; we donate to tech nonprofits.

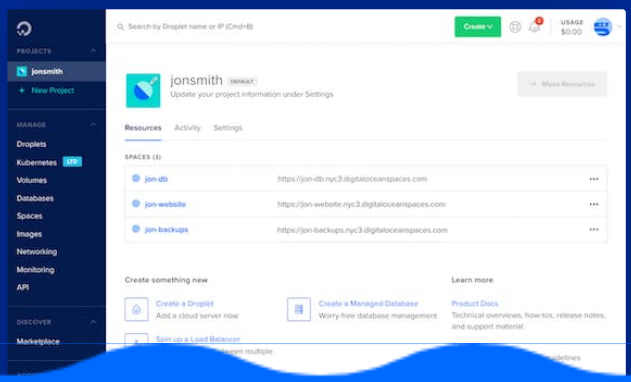
Featured on Community Kubernetes Course Learn Python 3 Machine Learning in Python Getting started with Go Intro to Kubernetes

DigitalOcean Products 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.

Learn More



Company

About
Leadership
Blog
Careers
Customers
Partners
Channel Partners
Referral Program
Affiliate Program
Press
Legal
Security
Investor Relations
DO Impact

Products

Products Overview
Droplets
Kubernetes
App Platform
Functions
Cloudways
Managed Databases
Spaces
Marketplace
Load Balancers
Block Storage
Tools & Integrations
API
Pricing
Documentation
Release Notes
Uptime

Community

Tutorials
Q&A
CSS-Tricks
Write for DOnations
Currents Research
Hatch Startup Program
deploy by DigitalOcean
Shop Swag
Research Program
Open Source
Code of Conduct
Newsletter Signup
Meetups

Solutions

Website Hosting
VPS Hosting
Web & Mobile Apps
Game Development
Streaming
VPN
SaaS Platforms
Cloud Hosting for Blockchain
Startup Resources

Contact

Support
Sales
Report Abuse
System Status
Share your ideas