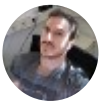




Published in CodeX

This is your **last** free member-only story this month.

[Sign up for Medium and get an extra one](#)



Ziga Petek

[Follow](#)

Aug 7, 2022 · 6 min read · · Listen



Save



# How to Use Nginx as a Reverse Proxy

## Build a Gate Keeper For Your Servers



Photo by [Waldemar Brandt](#) on [Unsplash](#)

Usually, when building a web application you do not build just one application. You have databases, APIs, messaging buses, and much more.

Access control to your services is crucial.

A reverse proxy is a perfect tool to handle that. And Nginx is the perfect tool to act as a reverse proxy. Here I'll show you how to set up Nginx as a reverse proxy on a Linux server.

**Imagine...**



Photo by [Dillon Kydd](#) on [Unsplash](#)

Imagine you have a big house. Let me rephrase it, imagine it's 1972 and you can afford a big house. Your house probably would have multiple doors, one main entrance, the garage door, the backdoor, and the door to the poolhouse,...

It would be pretty hard to control all of those doors so unwanted guests could not enter. So, what is the perfect solution to protect your house, and I mean to really protect it? You would build a wall around it. The wall would have only one door and at that door, you put a doorkeeper.



He would stay there, day and night, greet each of your guests and point them to where they needed to go. The mechanic would be able to get into the garage, the pool boy into the pool house, your visitors to the main entrance, and your family to the backdoor.

Welcome to the world of doorkeepers or, as us internet guys call it, reverse proxies.

A reverse proxy is an application, usually a web server, that accepts all incoming requests and forwards them to the right location. All of your other applications are only accessible through your reverse proxy.

This way all incoming requests can be managed through one central point and are thus way easier to manage.

There are many benefits of using a reverse proxy:

- as already mentioned you control all requests through a single entry point
- you can easily manage and redirect requests if needed.
- the reverse proxy can serve static content (such as pre-cached websites) and thus reduce the load on your applications
- the reverse proxy can act as an additional cache and thus further reduce load

All of that sounds pretty good. Now that we know the benefits of a reverse proxy, how do we set up one of those? It is relatively easy and we will use Nginx for that.

Nginx is a web server / reverse proxy / load balancer / mail proxy / HTTP cache. It was not designed by a swiss company, but it might as well be since many consider it to be the swiss knife of the web. Installing and configuring Nginx can be easy or complicated, depending on your needs. Setting it up as a reverse proxy is pretty easy though, don't worry.

## Our hypothetical server



Photo by [Jainath Ponnala](#) on [Unsplash](#)

A reverse proxy only makes sense if you have multiple applications that have to be accessed through the web. So let's define the theoretical server applications that we will need to access.

First, we have our website. It is a simple and nice NodeJS application, accessible through port 8080 and displaying your awesome portfolio. Users will be able to access it through the URL <http://my-awesome-portfolio-site.com>.

The website uses a database. You did not bother to implement a separate backend since you are so awesome and skilled that you can edit the content directly through the database. However, you will need access to it. Your database backend will locally be accessible through <http://127.0.0.1:1234> and you will be able to remotely access it through the URL <http://my-awesome-portfolio-site.com/db>

You have a lot of images, so you will need a media server. The local URL will be <http://localhost:2345>. It will be accessible through a separate URL: <http://media.my-awesome-portfolio-site.com>.

That's a neat little configuration we managed to set up there, isn't it? Naturally, you also set up a firewall to restrict access to all of your services from outside. Without a

reverse proxy, there is no way anyone will be able to access anything.

Now we have our server setup and we need a reverse proxy to manage access to all our services. Let's get to work.

## Get to work

First, install Nginx on your server:

```
1 sudo apt-get update -y
2 sudo apt-get upgrade -y
3 sudo apt-get install nginx -y
```

install\_nginx.sh hosted with ♥ by GitHub

[view raw](#)

The first two commands will update your server to the latest version. The last command installs Nginx. Use the '-y' flag to suppress the confirmation messages.

After successful installation, the server starts automatically. You can confirm that by checking the status of the server:

```
1 sudo systemctl status nginx
```

checkStatus.sh hosted with ♥ by GitHub

[view raw](#)

You should see something similar to this:

```
● nginx.service - nginx - high performance web server
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2022-07-19 20:31:19 CEST; 2 weeks 0 days ago
```

You can tell NginX what to do by changing the config files. You can usually find them in the conf.d folder: `"/etc/nginx/conf.d"`

Every ".conf" file will be used, so be careful what you put in this folder.

Since we don't need anything complex we can add just one configuration file named "myawesomesite.conf":

```
1  server {
2
3      listen 80;
4      listen [::]:80;
5
6      server_name my-awesome-portfolio-site.com;
7
8      location /db {
9          proxy_pass http://127.0.0.1:1234;
10     }
11
12     location / {
13         proxy_pass http://127.0.0.1:8080;
14     }
15 }
16
17 server {
18
19     listen 80;
20     listen [::]:80;
21
22     server_name media.my-awesome-portfolio-site.com;
23
24     location / {
25         proxy_pass http://127.0.0.1:2345;
26     }
27
28 }
```

reverseProxyExample.conf hosted with ❤ by GitHub

[view raw](#)

Let me go through each line:

### ***server {***

*This line indicates the start of a server block. Every configuration for a domain goes into such a block. As you can see we have two blocks, one for our main domain and one for our subdomain. The /db subdirectory goes into the main server block since it is technically a subfolder.*

### ***listen 80;***

### ***listen [::]:80;***

Both lines indicate that Nginx should listen to the port 80. The first line is for IPv4, the second is for IPv6.

*server\_name my-awesome-portfolio-site.com;*

the server name tells Nginx to which domains the server block should apply. In our case, we have two domains with different settings, thus two server blocks.

*location /db {*

This one tells Nginx for which paths the configuration between the brackets should apply. In our first server block, we need to forward all requests except for the requests to the db folder to our website. The */db* requests should go to our database backend.

That's why we have two location directives. In our second server block, we need to forward all requests to our media server, that's why we only have one */location* directive.

*proxy\_pass http://127.0.0.1:1234;*

This line is the most important and it's where the magic happens. It tells Nginx it should forward specific requests to an internal url. It does not have to be an internal URL, you can forward the requests to wherever you want. In our case, all services reside on the same server so internal URLs are enough.

Once you saved the configuration you need to reload Nginx:

```
1 sudo nginx -t
2 sudo systemctl reload nginx
```

nginxConfigReload.sh hosted with ♥ by GitHub

[view raw](#)

*sudo nginx -t* will check if the config is valid. If you did it right you should get something like this:

```
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

Nginx configuration test

And that's about it. I told you it would be easy. Happy coding!


## Related Articles



<p><b>How to Host a Website With a Dynamic IP And no Open Port</b></p> <p>In case your ISP is a jerk</p> <p>ziga-petek.medium.com</p>	
<p><b>How to Create a Home Network and Reverse Proxy with DnsMASq and Nginx</b></p> <p>Make Your Home Network Enjoyable</p> <p>ziga-petek.medium.com</p>	
<p><b>How to Point a Domain Name to Your Server</b></p> <p>So people will easier find you</p> <p>ziga-petek.medium.com</p>	
<p><b>How to Use Private/Public Key Authentication With SSH</b></p> <p>Forget your passwords on purpose!</p> <p>ziga-petek.medium.com</p>	



NginxReverse ProxyServersTrafficGate

Open in app ↗

Sign upSign In





 47 | 



Your tip will go to Ziga Petek through a third-party platform of their choice, letting them know you appreciate their story.

Give a tip

---

## Sign up for CrunchX

By CodeX

A weekly newsletter on what's going on around the tech and programming space [Take a look.](#)

Your email

---



Get this newsletter

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.

[About](#) [Help](#) [Terms](#) [Privacy](#)

Get the Medium app

