02 - Running a Web Server on a Droplet Using a Domain Name, Nginx, Certbot, PM2, & Express

DO Tutorial

Get a Domain & Point it at your Server's IP Address

When you create a droplet on DO, DO assigns it a static public IP address that uniquely identifies your machine among all the machines connected to the internet. We were able to use that IP address to ssh into our machine. If you kept a port open on the droplet's firewall & started a web server listening to requests on that port, you've got everything you need to start serving web pages from your server by going to *your_servers_ip_address:the_port_you_kept_open* in your browser. Unfortunately, IP addresses are pretty hard to remember. That's where a domain name comes in--it lets us create a friendly, human-readable name for our server's IP address.

1. Choose a domain registrar.
    - DO does not register new domains, though they do have tools for managing them, so you'll need to find another company to register a domain. If you don't know where to start, DO maintains this list of registrars that their tools are compatible with if you'd like to use DO to manage your domain.
2. Buy your domain through the registrar. This should be a pretty straightforward process, though choosing a relevant, available name might be a struggle!
3. Point your domain's A records at your server's IP address.
    - The first one will map *www.yourdomain.whatevz* to your server. Set *hostname* to "www".
    - The second one will map to *yourdomain.whatevz.* Instead of "www", put an "@" into the *hostname* field.

Install & Configure Nginx (or some other web server)

Nginx will make it really easy to (1) route requests coming into our server to the appropriate application & (2) snag a free SSL certificate to encrypt HTTP traffic to our little server which makes your site reachable via the coveted *https* scheme.

As usual, DO has an awesome tutorial for installing nginx which we're just going to follow along with here.

WARNING: The DO tutorial recommends creating server blocks to manage your Nginx config. We're going to skip that step for simplicity's sake though I'm not sure what the consequences of this shortcut will be later.

1. Use ssh to remote into your machine. If everything is set up correctly, you should be able to use your domain name instead of the droplet's IP address!
2. Install Nginx
    1. `sudo apt update`
    2. `sudo apt install nginx`
3. Adjust your firewall to enable access to Nginx. Thankfully, Nginx makes this really easy!
    1. Run `sudo ufw app list` to see some predefined firewall configurations *Nginx* makes available. Don't forget, *ufw* is the software firewall we're using.
    2. Start by allowing unencrypted HTTP traffic to Port 80 by running `sudo ufw allow 'Nginx HTTP'.`
    3. Verify the changes by running `sudo ufw status`. This should list our previous configuration allowing *ssh* & now *Nginx HTTP* as well.
4. Check that Nginx is running
    1. Run `systemctl status nginx`. Ubuntu automatically starts Nginx after it's installed, so this should return some messages indicating that Nginx is loaded & running.
    2. Back on your computer, try going to http://*your_servers_ip_address* in your browser. You should hit Nginx's default landing page.
5. Configure Nginx for your domain
    1. Open Nginx's default configuration file: `sudo nano /etc/nginx/sites-available/default`
    2. Find the line with *server_name*
    3. Replace the underscore with both versions of your domain name that we configured above: *www.yourdomain.whatevz yourdomain.whatevz*
    4. Confirm your configuration by running `sudo nginx -t`. Any errors? Confirm that there aren't any typos in the config file we just edited.

5. Reload Nginx: `sudo systemctl reload nginx`

Configure Nginx to Point to Your App

It's time to tell Nginx to route requests to our app!

1. Open up our default Nginx config file: `sudo nano /etc/nginx/sites-available/default`
2. Find the *location /* block and update it with the following. Make sure that the port number after *http://localhost:* matches the one configured in your express app.

```
  location / {
        proxy_pass http://localhost:3000;
        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;
  }
```

3. Save the file & check that it is still valid (those semi-colons are required) by running the following command again: `sudo nginx -t`
4. Reload Nginx: `sudo systemctl reload nginx`

Use PM2 to Run an Express App as a Daemon / Service

This part assumes you've deployed a working Express app to your machine & have already installed both node & npm on your droplet. A quick and dirty *git clone* to a public repo can do the trick to deploy your app to the droplet. When we run it from the terminal using a command like *npm start* from the app's directory, we're running it from a process that's completely dependent on our terminal session remaining open & not crashing for any reason. That's not feasible for a web server that you probably want to make available 24/7.

PM2 is a process manager that allows us to run node apps as a service--we can configure the app to run whenever our droplet turns on and to restart if it crashes for any reason.

1. Install PM2 as a global module
    - Run `sudo npm install pm2@latest -g`

2. Tell PM2 to run your app as a daemon / service
    - From your app's root directory, run `pm2 start nameOfYourFile.js`
    - OR, if you need to run an npm start script instead, run `pm2 start npm -- start`
3. Configure PM2 to run whenever the droplet starts up
    - run `pm2 startup systemd`
    - That command should output another command that you need to copy & paste & run with elevated permissions
4. Save your updated configuration with `pm2 save`

Secure Traffic to Your Nginx Server with an SSL Certificate

Here's the DO tutorial we'll be following for this step. As it explains, most of the process of setting up an SSL certificate can be automated if you choose *Let's Encrypt* as your Certificate Authority, the organization that creates your SSL certificate.

1. Install *Certbot*, the software tool created by *Let's Encrypt* to easily set up SSL certificates.
    - Add the Certbot repository: `sudo add-apt-repository ppa:certbot/certbot`
    - Install Certbot's Nginx package: `sudo apt install python-certbot-nginx`
2. Enable HTTPS traffic through your firewall:
    - `sudo ufw allow 'Nginx Full'`
    - `sudo ufw delete allow 'Nginx HTTP'` ('Nginx Full' will cover HTTP traffic, so let's keep our firewall rules clean)
3. Obtain your SSL certificates & configure Nginx with them in one fell swoop:
    - Run `sudo certbot --nginx -d yourdomain.whatevz -d www.yourdomain.whatevz`
    - You'll be prompted for an email address & to accept some Terms of Service.
    - You'll also be prompted whether you'd like to redirect all traffic to HTTPS or continue accepting traffic over HTTP. It's probably best to redirect everything to HTTPS.
4. Verify that Certbot can auto-renew your certs by running `sudo certbot renew --dry-run`

After all that work, you should be able to hit your app by going to https://yourdomain.whatevz!