Nginx

From NixOS Wiki
Jump to: navigation, search

Nginx (https://nginx.org/) (wikipedia:en:Nginx (https://en.wikipedia.org/wiki/en:Nginx)) is a lightweight webserver. Configuration is handled using the `services.nginx. (https://search.nixos.org/options?query=services.nginx.)` options.

Replace dependencies like openssl

In wake of the 2022 OpenSSL library, Nix can support in mitigating the library by downgrading (or replacing) the SSL library. For this, the overlay (/wiki/Overlay) facility of nixpkgs can be used:

```
nixpkgs.overlays = [
   (final: super: {
        nginxStable = super.nginxStable.override { openssl = super.pkgs.libressl; };
    } )
];
```

When utilizing NixOS options the following configuration will also work:

```
services.nginx.package = pkgs.nginxStable.override { openssl = pkgs.libressl; };
```

Let's Encrypt certificates

The nginx module for NixOS has native support for Let's Encrypt certificates; `services.nginx.+acme (https://search.nixos.org/options?query=services.nginx.+acme)` . The NixOS Manual, Chapter 20. SSL/TLS Certificates with ACME (https://nixos.org/nixos/manual/#module-security-acme-nginx) explains it in detail.

Minimal Example

Assuming that `myhost.org` resolves to the IP address of your host and port 80 and 443 has been opened.

```
services.nginx.enable = true;
services.nginx.virtualHosts."myhost.org" = {
    addSSL = true;
    enableACME = true;
    root = "/var/www/myhost.org";
};
security.acme = {
  acceptTerms = true;
  defaults.email = "foo@bar.com";
};
```

This will set up nginx to serve files for `myhost.org`, automatically request an ACME SSL Certificate and will configure systemd timers to renew the certificate if required.

Troubleshooting

Read-only Filesystem for nginx upgrade to 20.09

With the upgrade to nixos-20.09 the nginx comes with extra hardening parameters, most prominently the restriction of write access to the Operating System Disk. When you see errors like `[emerg] open() "/var/spool/nginx/logs/binaergewitter.access.log" failed (30: Read-only file system)` you can add extra paths to nginx service like this:

```
  systemd.services.nginx.serviceConfig.ReadWritePaths = [ "/var/spool/nginx/logs/" ];
```

SIGTERM received from 1

If you turn debug logging on:

```
services.nginx.logError = "stderr debug";
```

You may see this:

```
[notice] 12383#12383: signal 15 (SIGTERM) received from 1, exiting
```

This means systemd is killing nginx for you, but systemd (in nixOS 20.09) isn't nice enough to tell you why it's happening. Chances are it's because your nginx config has daemon mode turned on, turn off daemon mode in your nginx config like so:

```
daemon off;
```

And it should fix nginx so systemd won't go killing your nginx anymore.

Escape special chars in Regular Expressions

Some nginx configuration options like `locations` allows the use of Regular Expressions. Be ware that you need to escape some special chars (https://nixos.org/manual/nix/stable/language/values.html#type-string) like `\`, if provided by a double quoted `" "` string.

A common example found on the internet is:

```
locations."~ ^(.+\.php)(.*)$"  = {
    ...
};
```

But in this case the `\.php` part will be parsed by Nix to `.php`. In RegEx the dot represents any character instead of the dot character itself. Thus the path /gly**php**ro.css will be matched, too. Additionaly to the intended match of `/somephpfile.php?param=value`.

To circumvent this error `\.php` has to be double escaped as `\\.php`

```
locations."~ ^(.+\\.php)(.*)$"  = {
    ...
};
```

General

Nginx is run as the systemd service nginx, so `systemctl status nginx` may say something useful. If you have a problem with configuration, you can find the configuration location in the `systemctl status`, it should be at `/nix/store/*-nginx.conf`.

Sample setups

Static blog with ssl enforced in `configuration.nix`

```
services.nginx = {
  enable = true;
  virtualHosts."blog.example.com" = {
    enableACME = true;
    forceSSL = true;
    root = "/var/www/blog";
  };
};
# Optional: You can configure the email address used with Let's Encrypt.
# This way you get renewal reminders (automated by NixOS) as well as expiration emails.
security.acme.certs = {
  "blog.example.com".email = "youremail@address.com";
};
```

LEMP stack

(Nginx/MySQL/PHP) in `configuration.nix`

```
{ config, ...}: {
services.nginx = {
  enable = true;
  virtualHosts."blog.example.com" = {
    enableACME = true;
    forceSSL = true;
    root = "/var/www/blog";
    locations."~ \\.php$".extraConfig = ''
      fastcgi_pass  unix:${config.services.phpfpm.pools.mypool.socket};
      fastcgi_index index.php;
    '';
  };
};
services.mysql = {
  enable = true;
  package = pkgs.mariadb;
};
services.phpfpm.pools.mypool = {
  user = "nobody";
  settings = {
    "pm" = "dynamic";
    "listen.owner" = config.services.nginx.user;
    "pm.max_children" = 5;
    "pm.start_servers" = 2;
    "pm.min_spare_servers" = 1;
    "pm.max_spare_servers" = 3;
    "pm.max_requests" = 500;
  };
};
```

HTTP Authentication

Basic Authentication

Nginx can require users to login using HTTP Basic Authentication. In NixOS, this is set using the `basicAuth` option:

```
services.nginx = {
    virtualHosts."example.com" =  {
      basicAuth = { user = "password"; anotherUser = "..."; };
      ...
    };
};
```

Authentication via PAM

It is also possible to authenticate system users, e.g. users in the /etc/passwd file, by using the PAM module.

```
  security.pam.services.nginx.setEnvironment = false;
  systemd.services.nginx.serviceConfig = {
    SupplementaryGroups = [ "shadow" ];
  };

  services.nginx = {
    enable = true;
    additionalModules = [ pkgs.nginxModules.pam ];
    ...
    virtualHosts."example.com".extraConfig = ''
      auth_pam  "Password Required";
      auth_pam_service_name "nginx";
    '';
    ...
    };
  };
```

TLS reverse proxy

This is a "minimal" example in terms of security, see below for more tips.

```
services.nginx = {
    enable = true;
    recommendedProxySettings = true;
    recommendedTlsSettings = true;
    # other Nginx options
    virtualHosts."example.com" =  {
      enableACME = true;
      forceSSL = true;
      locations."/" = {
        proxyPass = "http://127.0.0.1:12345";
        proxyWebsockets = true; # needed if you need to use WebSocket
        extraConfig =
          # required when the target is also TLS server with multiple hosts
          "proxy_ssl_server_name on;" +
          # required when the server wants to use HTTP Authentication
          "proxy_pass_header Authorization;"
          ;
      };
    };
};
```

Note: ACME won't be able to authenticate your domain if ports 80 & 443 aren't open in your firewall.

Hardened setup with TLS and HSTS preloading

For testing your TLS configuration, you might want to visit [1] (https://www.ssllabs.com/ssltest/index.html). If you configured preloading and want to apply for being included in the preloading list, check out [2] (https://hstspreload.org/). Please read enough about preloading to understand the consequences, as it takes some effort to be removed from the list.

```
services.nginx = {
    enable = true;

    # Use recommended settings
    recommendedGzipSettings = true;
    recommendedOptimisation = true;
    recommendedProxySettings = true;
    recommendedTlsSettings = true;

    # Only allow PFS-enabled ciphers with AES256
    sslCiphers = "AES256+EECDH:AES256+EDH:!aNULL";

    appendHttpConfig = ''
      # Add HSTS header with preloading to HTTPS requests.
      # Adding this header to HTTP requests is discouraged
      map $scheme $hsts_header {
          https   "max-age=31536000; includeSubdomains; preload";
      }
      add_header Strict-Transport-Security $hsts_header;

      # Enable CSP for your services.
      #add_header Content-Security-Policy "script-src 'self'; object-src 'none'; base-uri 'none';" always;

      # Minimize information leaked to other domains
      add_header 'Referrer-Policy' 'origin-when-cross-origin';

      # Disable embedding as a frame
      add_header X-Frame-Options DENY;

      # Prevent injection of code in other mime types (XSS Attacks)
      add_header X-Content-Type-Options nosniff;

      # This might create errors
      proxy_cookie_path / "/; secure; HttpOnly; SameSite=strict";
    '';

    # Add any further config to match your needs, e.g.:
    virtualHosts = let
      base = locations: {
        inherit locations;

        forceSSL = true;
        enableACME = true;
      };
      proxy = port: base {
        "/".proxyPass = "http://127.0.0.1:" + toString(port) + "/";
      };
    in {
      # Define example.com as reverse-proxied service on 127.0.0.1:3000
      "example.com" = proxy 3000 // { default = true; };
    };
};
```

Using realIP when behind CloudFlare or other CDN

When Nginx is behind another proxy it won't know the true IP address of clients hitting it. It will then pass down those the proxy's IP address instead of the client IP address. By using the nginx realip module, we can ensure nginx knows the real client IP, and we can further inform nginx to only trust the HTTP header from valid upstream proxies.

In the following example, we are fetching the list of IPs directly from cloudflare and including a hash. This has some pros and cons. Nix will not attempt to download or update that file while it is in a nix store it trusts, but after a nix garbage collection, it will error if the list of proxies has changed informing you of that when you apply the config.

```
  services.nginx.commonHttpConfig =
    let
      realIpsFromList = lib.strings.concatMapStringsSep "\n" (x: "set_real_ip_from  ${x};");
      fileToList = x: lib.strings.splitString "\n" (builtins.readFile x);
      cfipv4 = fileToList (pkgs.fetchurl {
        url = "https://www.cloudflare.com/ips-v4";
        sha256 = "0ywy9sg7spafi3gm9q5wb59lbiq0swvf0q3iazl0maq1pj1nsb7h";
      });
      cfipv6 = fileToList (pkgs.fetchurl {
        url = "https://www.cloudflare.com/ips-v6";
        sha256 = "1ad09hijignj6zlqvdjxv7rjj8567z357zfavv201b9vx3ikk7cy";
      });
    in
    ''
      ${realIpsFromList cfipv4}
      ${realIpsFromList cfipv6}
      real_ip_header CF-Connecting-IP;
    '';
```

UNIX socket reverse proxy

In order for nginx to be able to access UNIX sockets, you have to do some permission modifications.

```
# Example service that supports listening to UNIX sockets
services.hedgedoc = {
  enable = true;
  settings.path = "/run/hedgedoc/hedgedoc.sock"
};

services.nginx = {
  enable = true;
  virtualHosts."example.com" = {
    enableACME = true;
    forceSSL = true;
    locations."/".proxyPass = "http://unix:/run/hedgedoc/hedgedoc.sock";
  };
};

# This is needed for nginx to be able to read other processes
# directories in `/run`. Else it will fail with (13: Permission denied)
systemd.services.nginx.serviceConfig.ProtectHome = false;

# Most services will create sockets with 660 permissions.
# This means you have to add nginx to their group.
users.groups.hedgedoc.members = [ "nginx" ];

# Alternatively, you can try to force the unit to create the socket with
# different permissions, if you have a reason for not wanting to add nginx
# to their group. This might not work, depending on how the program sets
# its permissions for the socket.
systemd.services.hedgedoc.serviceConfig.UMask = "0000";
```

See more

- Official Documentation (http://nginx.org/en/docs/)

Categories (/wiki/Special:Categories): Applications (/wiki/Category:Applications)
Server (/wiki/Category:Server)