static websites with nginx

```nix
{inputs, ...}: {
  networking.firewall.allowedTCPPorts = [ 80 443 ];
  services.nginx = {
    enable = true;
    recommendedProxySettings = true;
    recommendedTlsSettings = true;
    virtualHosts = {
      "shobu.fr" = {
        enableACME = true;
        forceSSL = true;

        root = "${inputs.shoblog-front.packages.x86_64-linux.default}/dist";
      };
    };
  };
  security.acme = {
    acceptTerms = true;
    defaults.email = "lelu.awen@proton.me";
  };
}
```

On se retrouve pour la suite de notre introduction à **nix** et **nixos**, j'essaye un nouveau format ainsi que de rajouter un glossaire à la fin, aujourd'hui on parle de la partie serveur et déploiement avec nginx, alors comme ont dis au States: buckle up bookaroo.

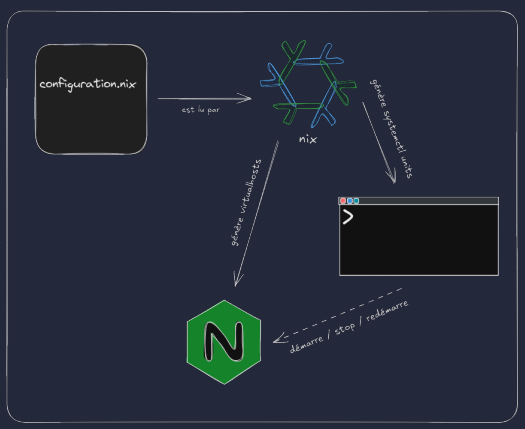
album du jour

once in a long long while - low roar

introduction

```nix
{inputs, ...}: {
  networking.firewall.allowedTCPPorts = [ 80 443 ];
  services.nginx = {
    enable = true;
    recommendedProxySettings = true;
    recommendedTlsSettings = true;
    virtualHosts = {
      "shobu.fr" = {
        enableACME = true;
        forceSSL = true;

        root = "${inputs.shoblog-front.packages.x86_64-linux.default}/dist";
      };
    };
  };
  security.acme = {
    acceptTerms = true;
    defaults.email = "lelu.awen@proton.me";
  };
}
```

Sous **nixos**, il est possible de définir la configuration réseau du système de façon déclarative, ainsi que des services, qui sont généralement des daemons que l'on peut configurer via nix.
Sous le capot, cette configuration va être interprétée par **nix** en des options de plus en plus bas niveau, jusqu'à générer les commandes **bash** et les **fichiers de configurations** nécessaires à l'application de la configuration demandée.
Ainsi, ultimement, cette configuration va nous générer un fichier de configuration **nginx** correspondant à notre virtualhost, ainsi que les commandes nécessaires pour lancer nginx (et l'installer évidemment)



Paramètres généraux

```
{inputs, ...}: {
  networking.firewall.allowedTCPPorts = [ 80 443 ];
  services.nginx = {
    enable = true;
    recommendedProxySettings = true;
    recommendedTlsSettings = true;
    virtualHosts = {
      "shobu.fr" = {
        enableACME = true;
        forceSSL = true;

        root = "${inputs.shoblog-front.packages.x86_64-linux.default}/dist";
      };
    };
  };
  security.acme = {
    acceptTerms = true;
    defaults.email = "lelu.awen@proton.me";
  };
}
```

Sous nixos, la configuration du système est composée de modules, chaque module est une fonction prenant un certain nombre de paramètres et retournant un set d'options décrivant la configuration de notre système. Au moment de builder cette dernière, tous les modules sont fusionnés afin de donner la configuration finale à appliquer, puis cette dernière est interprétée par nix.

```
{inputs, ...}: {
  # ...
}
```

On commence notre module en définissant les arguments d'entrée de ce dernier, les `...` agissent comme une wildcard, nous autorisant à fournir plus d'argument d'entrée dans l'appel de la fonction que ce que cette dernière ne vas actuellement utiliser. Ce pattern permet à nix de laisser au module la liberté de déclarer uniquement les dépendances dont il a besoin.
Ici, on déclare un argument `inputs` dont la déclaration est en dehors du scope de cet article, mais qui contient les dépendances de notre configuration, incluant le site packager dans l'article précédent.

```
security.acme = {
  acceptTerms = true;
  defaults.email = "lelu.awen@proton.me";
};
```

l'option `services.nginx` de nixos permet d'automatiquement manager l'obtention et le renouvellement des **certificats ssl**, par défaut, **let's encrypt** est utilisé, mais n'importe quel CA peut être utiliser pourvu qu'il supporte le protocole ACME. Afin d'utiliser ce service, il est nécessaire d'en accepter les conditions d'utilisations ainsi que de fournir un email.

Paramètres de Nginx

```nix
{inputs, ...}: {
  networking.firewall.allowedTCPPorts = [ 80 443 ];
  services.nginx = {
    enable = true;
    recommendedProxySettings = true;
    recommendedTlsSettings = true;
    virtualHosts = {
      "shobu.fr" = {
        enableACME = true;
        forceSSL = true;

        root = "${inputs.shoblog-front.packages.x86_64-linux.default}/dist";
      };
    };
  };
  security.acme = {
    acceptTerms = true;
    defaults.email = "lelu.awen@proton.me";
  };
}
```

Par défaut, nginx va écouter sur les ports 80 (pour les requètes http) et 443 (pour SSL), on dit donc au firewall d'ouvrir les ports correspondants.
On active ensuite le service Nginx de nixos, qui vas se charger d'installer la dernière version et de créer les units systemctl nécessaire à son fonctionnement. Ce service propose un bon paquet d'options, mais on va ici voir uniquement la base.
Les options `recommendedTlsSettings` et `recommendedProxySettings` vont automatiquement ajouter un set de paramètre à la conf générée de chaque virtualhost, à moins que cette option ne soit set à la négative au niveau du dit host.

Paramètres du virtualhost

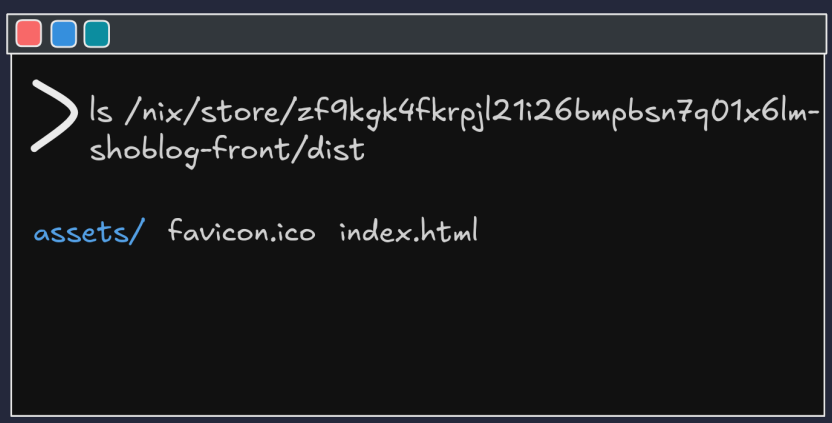
```nix
{inputs, ...}: {
  networking.firewall.allowedTCPPorts = [ 80 443 ];
  services.nginx = {
    enable = true;
    recommendedProxySettings = true;
    recommendedTlsSettings = true;
    virtualHosts = {
      "shobu.fr" = {
        enableACME = true;
        forceSSL = true;

        root = "${inputs.shoblog-front.packages.x86_64-linux.default}/dist";
      };
    };
  };
  security.acme = {
    acceptTerms = true;
    defaults.email = "lelu.awen@proton.me";
  };
}
```

Et voici l'option qui nous intéresse ici, cette option est un genre de dictionnaire, chaque clé correspond à un nom de domaine sur lequel notre virtualhost sera mapper, et la configuration propre à ce dernier. Voyons ça en détail:

`enableACME` dit à nixos que l'on souhaite utiliser le module ACME pour ce virtualHost, en utilisant ce dernier, il vas aller demander un certificat à let's encrypt qui correspond au domaine de notre vhost, et renouveler ce dernier régulièrement avant chaque expiration. De plus, il va aussi ajouter la configuration néscéssaire pour que notre vhost prenne en charge les requêtes HTTPS.

`forceSSL` force notre vhost à rediriger les requêtes http vers le https.

`root` définis le dossier que notre serveur va servir, et si ce dernier contient un fichier `index.html`, c'est ce fichier qui va être servis par défaut. L'astuce ici est d'utiliser directement le paquet composé de notre site packager précédement, après avoir pris soins d'ajouter celui-ci aux inputs de notre configuration, il se retrouve accessible via l'argument `input.<nom du paquet>.packages.<architecture>.default`. Au moment de l'interprétation du fichier par nix, cette variable va interpoler avec le chemin du paquet dans le nix store (un truc du genre `/nix/store/zf9kgk4fkrpjl21i26bmpbsn7q01x6lm-shoblog-front`), qui contient tout nos assets statiques.



Et voilà ! Un petit `nixos-rebuild switch` et le système lire notre config, télécharger les paquets manquants, builder notre site, copier les assets dans le store, créer notre virtualhost et demander les certificats. Et le plus beau c'est que si l'on met à jour notre application, il nous suffit juste de pousser les modifications sur notre repo et de dire à nix de checker si il y a eu des mises à jour dans ses inputs et ce dernier vas checker le repo pour des nouveaux comits sur la branche qu'il surveille et, le cas avenant, pull la nouvelle version, la build, et relancer les services correspondant comme un grand.

Glossaire

nixos: distribution linux à la configuration déclarative basée sur le package mnager **nix**
nix: package manager servant à définir des environnements logiciels de manière déclarative et reproductible
daemon: logiciel fonctionnement en arrière-plan, de façon transparente à l'utilisateur.
bash: interpréteur de commande pour les OS de type UNIX permettant d'interagir directement avec ce dernier.
nginx: serveur web et reverse proxy assez commun et open source.
virtualhost: dans le contexte de nginx, ensemble de la configuration associée à un domaine.
certificat ssl: fichier attestant de l'identité d'un domaine.
Certificat **A**uthority: Organisation fournissant des certificats aux propriétaires d'un domaine et servant de garantie à la validité de ces derniers.