SERVEURS WEB

SOMMAIRE

- QU'EST-CE QU'UN SERVEUR WEB ?
- RAPPELS SUR DOCKER
- APACHE HTTPD
- NGINX
- OPTIMISATION ET UTILISATION AVANCÉE
- CERTIFICATS SSL
- TÂCHES AUTOMATIQUES ET MISE EN CACHE

AVANT TOUTE CHOSE

- Machine virtuelle
  - https://www.virtualbox.org/
- Pack d'extensions
- Création d'une machine virtuelle
- Installation d'Ubuntu Server
  - https://releases.ubuntu.com/20.04.2/ubuntu-20.04.2-live-server-amd64.iso
- Configuration du NAT et de la redirection des ports

QU'EST-CE QU'UN SERVEUR WEB ?

QU'EST-CE QU'UN SERVEUR WEB ?

*“ Un **serveur web** est un logiciel qui va servir des ressources web en répondant aux requêtes HTTP de ses clients.*

*Cela peut également représenter la partie **matérielle** d'un serveur*

QU'EST-CE QU'UN SERVEUR WEB ?

“ ***HTTP : HyperText Transfer Protocol***, *protocole de la couche* ***applicative*** *collaboratif permettant la* ***distribution*** *d'informations* ***hypermédias***

QU'EST-CE QU'UN SERVEUR WEB ?

Tim Berners-Lee

- Informaticien britannique
- Protocole HTTP
- WWW / W3C
- Prix Turing 2016

QU'EST-CE QU'UN SERVEUR WEB ?

*“ Un **"user agent"** est un logiciel agissant pour l'utilisateur pour récupérer, interpréter et faciliter **l'interaction** avec le contenu web. Souvent le **client** dans un échange client / server. Exemple : client mail, navigateur web, ...*

Exemples de chaîne de caractères d'identification

```
Mozilla/5.0 (iPad; U; CPU OS 3_2_1 like Mac OS X; en-us) AppleWebKit/531.21.10
(KHTML, like Gecko) Mobile/7B405

Googlebot/2.1 (+http://www.google.com/bot.html)
```

QU'EST-CE QU'UN SERVEUR WEB ?

*“ Une **requête HTTP** est un message envoyé dans un certain format compris par le client et le serveur pour communiquer entre eux.*

Exemples de requêtes HTTP

```
http://www.example.com/path/file.html

GET /path/file.html HTTP/1.1
Host: www.example.com
```

QU'EST-CE QU'UN SERVEUR WEB ?

Requête HTTP

- ligne principale de la requête (informations diverses)
- champs d'en-têtes (*headers*)
- une ligne vide
- le contenu ou corps optionnel de la requête (*body*)

```
POST /auth/login HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

name=toto&password=nLBDfL4E4zVavaXP
```

QU'EST-CE QU'UN SERVEUR WEB ?

Méthodes HTTP

- **GET** : récupération simple de données avec représentation
- **HEAD** : similaire à **GET** mais sans corps de réponse
- **POST** : indique au serveur de prendre en compte les données postées
- **PUT** : similaire à **POST** mais indique une création ou une modification
- **DELETE** : indique au serveur de supprimer la ressource
- **OPTIONS** : récupération des méthodes supportées et d'infos
- **PATCH** : similaire à **PUT** mais en modification partielle
- **TRACE** : récupération des changements intermédiaires
- **CONNECT** : convertit la requête en tunnel TCP/IP

QU'EST-CE QU'UN SERVEUR WEB ?

Réponse HTTP

- globalement la même chose que pour la requête
- seule différence : la première ligne comprend le **code de statut** ainsi que la **raison**

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 155
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Connection: close

User created!
```

QU'EST-CE QU'UN SERVEUR WEB ?

Code de statut HTTP

- résume ce qu'il s'est passé après la requête faite au serveur
- convention pour tout développement d'une API par exemple
- différentes catégories existent en fonction des cas :
  - **10X** : retour d'informations
  - **20X** : la requête est un succès
  - **30X** : indique une redirection
  - **40X** : le client a fait une erreur
  - **50X** : une erreur est survenue côté serveur

QU'EST-CE QU'UN SERVEUR WEB ?

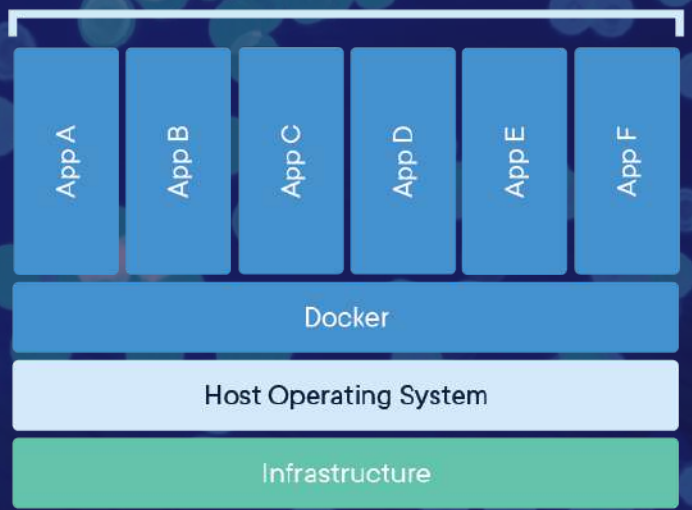
Fonctionnalités d'un serveur web

- support d'une ou plusieurs versions d'HTTP
- récupération des événements (logs)
- authentification pour accéder à une ressource
- gestion de gros fichiers
- limitation de la bande passante (bandwidth throttling)
- hébergement virtuel (vhost)
- modules et extensions pour ajouter d'autres fonctionnalités comme de l'interprétation de script côté serveur (PHP, Ruby, Python, ...)

QU'EST-CE QU'UN SERVEUR WEB ?

Quelques serveurs web

RAPPELS SUR DOCKER



- Gestion de conteneurs
- Différents des machines virtuelles
- Permet d'isoler des applications
- Plein d'images différentes

RAPPELS SUR DOCKER

Débuter avec Docker

- Création de son compte Docker Hub
- Installation de Docker Desktop
- Quelques commandes

```
docker run hello-world
docker images
docker ps -a
docker rm hello-world; docker rmi hello-world
```

RAPPELS SUR DOCKER

Exemple avec Portainer (application pour Docker)

- Linux / macOS

```
1 docker volume create portainer_data
2 docker run -d -p 8000:8000 -p 9000:9000 --name=portainer --
  restart=always -v /var/run/docker.sock:/var/run/docker.sock -v
  portainer_data:/data portainer/portainer-ce
```

- Windows

```
1 docker volume create portainer_data
2 docker run -d -p 8000:8000 -p 9000:9000 --name portainer --restart
  always -v \\.\pipe\docker_engine:\\.\pipe\docker_engine -v
  portainer_data:C:\data portainer/portainer-ce
```

RAPPELS SUR DOCKER

Docker compose

```yaml
version: '3.3'

services:
   db:
     image: mysql:5.7
     volumes:
       - db_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: somewordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress

   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "8000:80"
     restart: always
     environment:
```

APACHE HTTPD

APACHE HTTPD

- Serveur Web par Apache
- Gratuit, open-source, cross-platform
- Modulaire : authentification, langages multiples, proxy, ssl, réécriture, etc.

APACHE HTTPD

Utilisation basique avec Docker

- Création d'un dossier de test dans lequel on se place
- Via Docker

```
docker run -dit --name my-apache -p 8080:80 -v "$PWD":/usr/local/apache2/htdocs/
httpd:2.4
```

- Ici on utilise *$PWD* pour spécifier le dossier courant depuis lequel on exécute la commande
- On peut accéder à une page web sur http://localhost:8080

APACHE HTTPD

Utilisation basique avec Docker

- Très rapide à mettre en place mais pas idéal sur plein d'aspect : sécurité, modularité, etc.
- Ce fonctionnement avec Docker reste pour rapidement servir un site, lors du développement par exemple
- Il faut donc changer l'approche que l'on a avec une configuration adéquat à notre besoin

APACHE HTTPD

Que se passe t-il au lancement du serveur ?

- Chargement des fichiers de configurations
- Attente de requêtes sur les interfaces réseaux

> “ ***Écoute*** *du serveur*

Que se passe t-il quand on ouvre le site sur son navigateur ?

- Résolution du nom de domaine
- Requête HTTP en GET
- Réception par le serveur, interprétation et renvoi du contenu
- Traitement par le navigateur et affichage

APACHE HTTPD

Mise en place basique sur un serveur

```
sudo apt-get install apache2
```

On le manipule grâce à un service :

```
sudo systemctl enable|disable|start|stop|restart|reload apache2
```

Si systemd n'est pas disponible

```
sudo service apache2 enable|disable|start|stop|restart|reload
```

APACHE HTTPD

Mise en place basique sur un serveur

- En accédant à http://localhost:8080 une page web s'affiche
  - **Rappel** : on utilise le port 8080 car il pointe sur le port 80 de la machine virtuelle
  - Le port 80 est celui par défaut défini dans la configuration
- La page est disponible dans le dossier **/var/www/html**
  - C'est le dossier par défaut ciblé dans la configuration

APACHE HTTPD

Mise en place basique sur un serveur

- La configuration globale ne se modifie pas (normalement)
- On utilise plutôt une configuration morcelée :
  - **sites-xxx** : pour les sites
  - **conf-xxx** : pour les autres services
  - **mods-xxx** : pour les modules
- La différence **available** / **enabled** réside dans l'utilisation de liens symboliques pour activer une configuration ou non

APACHE HTTPD

Configuration du serveur

- Les fichiers de configuration doivent avoir l'extension **.conf**
  - Il faut rafraichir la configuration après modification
- La configuration pour les sites est lue dans cet ordre :
  - configuration globale : **apache2.conf**
  - configuration individuelle : **sites-enabled/*.conf**
    - Ici par défaut, on utilise le fichier **000-default.conf**
    - L'ordre alphabétique est respecté
- Le fichier **envvars** définit les variables d'environnement

APACHE HTTPD

Configuration du serveur

- **ServerRoot** : dossier principal du serveur (config, logs, etc.)
- **TimeOut** : nombre de secondes avant de faire échouer une requête trop longue
- **User** / **Group** : utilisateur et groupe Unix pour l'exécution
- **KeepAlive*** : permet d'activer la gestion optimisé de requêtes multiples via la même connexion TCP
- **ErrorLog** : dossier pour les logs d'erreur
- **Include*** : permet d'inclure un fichier de configuration
- **ServerName** : nom d'hôte et port d'identification du serveur
- **DocumentRoot** : dossier depuis lequel sont servis les fichiers

APACHE HTTPD

Configuration spécifique

- **<Directory ...>** : restreint des directives à un seul dossier, ses sous-dossiers et leurs fichiers correspondants
- **<FilesMatch ...>** : restreint des directives à des fichiers via une expression régulière
- **<VirtualHost ...>** : restreint des directives à un nom de domaine ou une adresse IP
- **<If ...>**, **<Else>** ... : exécute des directives selon une condition

APACHE HTTPD

Exercice

- Rendre accessible un site simple via le serveur web avec les règles suivantes :
  - Les fichiers du site doivent être placés dans **/srv/mysite**
  - L'hôte virtuel doit se trouver dans un nouveau fichier de configuration
  - Le site doit être accessible sur le port 8081
  - Le site doit également être accessible via le nom de domaine suivant : **myawesomesite.com**

APACHE HTTPD

Quels sont les problèmes ?

- Assez rapide à mettre en place mais pas du tout sécurisé :o
  - Presque tous les fichiers du dossier sont accessibles
  - Exemple avec un fichier *secret.txt* dans le dossier
- Dans l'idéal il faut séparer le code du site, des assets, des fichiers de configuration, etc.
- La configuration suffit pour un site statique mais qu'en est-il d'un site en PHP par exemple ?

APACHE HTTPD

Distribution d'un site plus élaboré

- Séparation du code, des assets et de la configuration
  - **assets/** pour les images, les feuilles de styles, etc.
  - **pages/** pour les pages HTML
- Changement d'option d'accès de certains fichiers
- Sécurisation de pages avec une authentification basique
- Changement de la sortie des logs
- Ajout d'une page d'erreur spécifique
- Ajout d'un système de réécriture d'URL

APACHE HTTPD

Changement des fonctionnalités d'un scope

- Utilisation de la directive **Options** avec différents arguments
  - **All** : activation de toutes les options (sauf **MultiViews**)
  - **FollowSymLinks** : permet de suivre les liens symboliques
  - **Indexes** : liste les fichiers du dossier s'il manque l'index
  - **MultiViews** : permet de filtrer plus précisément la distribution

```
Options Indexes FollowSymLinks
```

APACHE HTTPD

URL pour cibler le système de fichier

- **Alias** : fichiers situés en dehors du **DocumentRoot**
  - chemin de l'URL suivi du dossier ciblé
  - on peut aussi simplement utiliser les liens symboliques
- **AliasMatch** : permet l'utilisation d'expressions rationnelles

```
Alias /path /var/www/othersite
AliasMatch "^/~([a-zA-Z0-9]+)/mypath/(.+)" "/home/$1/mydocs/$2"
```

- **DirectoryIndex** : permet de définir le fichier par défaut

APACHE HTTPD

Utilisation du fichier .htaccess

- Le fichier **.htaccess** permet de définir des directives par rapport au dossier dans lequel il se trouve
  - Pour configurer ce qu'il est autorisé de modifier on utilise l'instruction **AllowOverride** en précisant le groupe (ligne **Override** dans la documentation)
- Il ne devrait être utilisé que si l'on a pas accès à la configuration principale

```
AllowOverride AuthConfig
```

APACHE HTTPD

Authentification

- L'accès à certaines ressources peut être décidé via différents types d'authentification
  - **None** : ne définit pas d'authentification
  - **Basic** : authentification basique via utilisateur / mdp
  - **Digest** : authentification basée sur les condensés MD5
  - **Form** : authentification utilisant les cookies HTTP
- Modules : **mod_authn_core** et **mod_authz_core**
- La directive **Require** permet de moduler aussi les accès

APACHE HTTPD

Authentification basique

- **AuthType** : Basic
  - Nécessité de générer un fichier de mot de passe avec l'utilitaire **htpasswd**
- **AuthName** : chaîne d'identification de la zone restreinte
- **AuthUserFile** : fichier de mot de passe
- **Require** : défini ici les utilisateurs autorisés

```
AuthType Basic
AuthName "Restricted Area"
AuthUserFile "/etc/apache2/passwd/passwords"
Require user rkiffer
```

APACHE HTTPD

Authentification basique

- Il existe également d'autres fournisseurs pour gérer l'authentification autrement qui via un fichier :
  - **AuthBasicProvider** :
    - **file** : par défaut, pour lire un fichier de mots de passe
    - **dbm** : pour lire dans un fichier DBM
    - **dbd** : pour lire dans une base SQL
    - **ldap** : pour lire dans un annuaire LDAP
- On peut aussi utiliser plusieurs fournisseurs en même temps

APACHE HTTPD

Reverse Proxy (Mandataire Inverse)

- Permet de récupérer des ressources servies en local sur le serveur via d'autres URL
- Se positionne devant le client pour simuler leur provenance
  - **ProxyPass** : configure le rapatriement
  - **ProxyPassReverse** : réécrit les redirections
- On peut utiliser **Substitute** pour réécrire des liens via Sed

```
ProxyPass "/foo/" "http://internal.example.com/bar/"
ProxyPassReverse "/foo/" "http://internal.example.com/bar/"
ProxyPassReverseCookieDomain internal.example.com public.example.com
ProxyPassReverseCookiePath "/foo/" "/bar/"
```

APACHE HTTPD

> ***Tu es un sorcier Harry !***

- Module : **mod_rewrite** (à activer)
- Permet quasiment tout type de réécriture d'URL
- Assez complexe à prendre en main
- Utilisation des expressions rationnelles
- Attention au contexte d'utilisation :
  - Scope, fichier .htaccess, ...
- Il est conseillé de bien configurer ses logs

```
LogLevel rewrite:trace6
```

APACHE HTTPD

Réécriture d'URL

- Plusieurs instructions importantes :
  - **RewriteEngine** : active ou désactive le module
  - **RewriteCond** : condition de réécriture
  - **RewriteRule** : règle de réécriture
  - **RewriteBase** : base de la réécriture de répertoire
- Il existe aussi beaucoup de variables à utiliser pour récupérer certaines informations du serveur, de la requête, etc.
  - **%{REQUEST_URI}** : chemin de l'URI de la requête
  - **%{REMOTE_ADDR}** : adresse IP de l'hôte distant

APACHE HTTPD

Réécriture d'URL - Règle

- Une règle est constitué de 3 parties :
  - **Modèle** : identifie les URL à qui appliquer la règle via une expression régulière appliquée au chemin de l'URL
  - **Substitution** : définit la transformation à appliquer
    - chemin complet du système vers une ressource (alias)
    - chemin web vers une ressource
    - une URL absolue
  - **[drapeaux]** : options modifiant le comportement

APACHE HTTPD

Réécriture d'URL - Condition

- Restreint la règle de réécriture définie à la suite
- Une condition est constitué de 3 parties :
  - **Variable** : chaîne à tester par la condition
  - **Condition** : expression rationnelle devant correspondre à la variable précédente
  - **[drapeaux]** : options modifiant l'évaluation de la condition
- Il est possible d'avoir plusieurs conditions avant la règle

APACHE HTTPD

Exercice

- Distribuer la nouvelle version du site avec ces règles :
  - Les feuilles de styles doivent être accessibles via **/styles**
  - Les images doivent être accessibles via **/images**
  - La page **/calendar** doit être accessible uniquement via une authentification basique
  - Les logs doivent être positionnés dans un autre dossier
  - Les fichiers ne doivent pas être indexables
  - Rendre accessibles les pages sans l'extension **.html**
  - Afficher la page 404.html en cas d'erreur 404

APACHE HTTPD

PHP et FastCGI

- Historiquement, jusqu'à la version 2.4 d'Apache il fallait utiliser le module **mod_php**
- Utilisation seulement en mode **prefork**
  - Problème de performances avec une forte charge
- Pas de possibilité d'utiliser HTTP/2
- Rien d'insurmontable mais cela nécessitait une bonne optimisation de son utilisation

APACHE HTTPD

PHP-FPM : le renouveau !

- **PHP-FPM** : gestionnaire de processus FastCGI pour PHP
  - On peut l'utiliser avec le module **MPM Event** d'Apache pour gérer plus de connexion en consommant moins

```
sudo apt install php-fpm
service php7.4-fpm start
```

- Il faut aussi un moyen pour Apache et PHP de communiquer

```
sudo apt install libapache2-mod-fcgid
```

APACHE HTTPD

PHP-FPM : le renouveau !

- Il faut ensuite activer la configuration pour la communication ainsi que les modules nécessaires

```
sudo a2enconf php7.4-fpm
sudo a2enmod proxy
sudo a2enmod proxy_fcgi
```

- Pour tester, il suffit de créer un fichier **info.php** avec le contenu suivant dans le dossier **/var/www/html**

```
<?php phpinfo(); ?>
```

APACHE HTTPD

PHP-FPM : le renouveau !

- L'installation de ces différents modules et dépendances a ajouté des configurations pour interpréter les fichiers PHP

```
<FilesMatch ".+\.ph(ar|p|tml)$">
        SetHandler "proxy:unix:/run/php/php7.4-fpm.sock|fcgi://localhost"
</FilesMatch>
<FilesMatch ".+\.phps$">
        # Deny access to raw php sources by default
        # To re-enable it's recommended to enable access to the files
        # only in specific virtual host or directory
        Require all denied
</FilesMatch>
# Deny access to files without filename (e.g. '.php')
<FilesMatch "^\.ph(ar|p|ps|tml)$">
        Require all denied
</FilesMatch>
```

NGNIX

NGINX

- Serveur Web par NGINX, Inc. et Igor Sysoev
- Gratuit sous licence BSD, deuxième serveur mondial
- Serveur web, reverse-proxy, load balancer, proxy mail, cache HTTP, etc.

NGINX

Utilisation basique avec Docker

- Création d'un dossier de test dans lequel on se place
- Via Docker

```
docker run -d --name my-nginx -p 8080:80 -v "$PWD":/usr/share/nginx/html
nginx:1.19
```

- Ici on utilise *$PWD* pour spécifier le dossier courant depuis lequel on exécute la commande
- On peut accéder à une page web sur http://localhost:8080

NGINX

Mise en place basique sur un serveur

```
sudo apt-get install nginx
```

On le manipule grâce à un service :

```
sudo systemctl start|stop|restart|reload|force-reload|status nginx
```

Si systemd n'est pas disponible

```
sudo service nginx start|stop|restart|reload|force-reload|status
```

NGINX

Mise en place basique sur un serveur

- En accédant à http://localhost une page web s'affiche
  - Le port 80 est celui par défaut défini dans la configuration
- La page est disponible dans le dossier **/var/www/html**
  - C'est le dossier par défaut ciblé dans la configuration
- Pour le moment on se rapproche de la configuration de base d'Apache sauf pour la fonctionnalité d'indexation

NGINX

Mise en place basique sur un serveur

- La configuration se trouve dans **/etc/nginx/nginx.conf**
- Comme Apache HTTPD, la configuration est séparée :
  - **sites-xxx** : pour les sites
  - **modules-xxx** : pour les modules
  - **conf.d** : pour les autres configurations
- La différence **available** / **enabled** réside dans l'utilisation de liens symboliques pour activer une configuration ou non

NGINX

Configuration du serveur

- Les fichiers de configuration n'ont pas forcément d'extension
  - Il faut rafraichir la configuration après modification
- La configuration pour les sites est lue dans cet ordre :
  - configuration globale : **nginx.conf**
  - configurations des modules : **modules-enabled/*.conf**
  - configurations diverses : **conf.d/*.conf**
  - configurations individuelles : **sites-enabled/***
    - Ici par défaut, on utilise le fichier **default**
- Sans oublier : **mime.types**, ***_params**, etc.

NGINX

Configuration du serveur

- **user** : utilisateur des processus de travail (groupe aussi)
- **worker_processes** : nombre de processus de travail
- **include** : inclusion d'autres fichiers de configuration
- **events** {} : contexte de traitement des connexions
  - **worker_connections** : connexions simultanées max
- **http** {} : contexte du serveur HTTP
  - **gzip** : activation de la compression des réponses
- **server** {} : déclaration d'un serveur virtuel (vhost)
  - **server_name** : nom de domaine du serveur
  - **listen** : adresse et/ou port d'écoute des requêtes

NGINX

Serveurs virtuels

- Permet d'héberger plusieurs site web sur une même machine
- Possibilité de préciser une IP, un port, un nom de domaine
- On peut configurer chaque serveur virtuel indépendamment
  - **default_server** : identifie le serveur par défaut (listen)
  - **root** : dossier racine pour les requêtes
  - **index** : fichier correspondant à l'index (point d'entrée)
  - **location** {} : contexte par rapport à une URI
  - **try_files** : sert le premier fichier de la liste ou le fallback

NGINX

Exercice (same ** all over again...)**

- Rendre accessible un site simple via le serveur web avec les règles suivantes :
  - Les fichiers du site doivent être placés dans **/srv/mysite**
  - L'hôte virtuel doit se trouver dans un nouveau fichier de configuration
  - Le site doit être accessible sur le port 8081
  - Le site doit également être accessible via le nom de domaine suivant : **myawesomesite.com**

NGINX

Alias et redirections

- Il faut se placer dans un contexte **location**
- Pour un alias, deux comportements différents :
  - **root** : vers le chemin spécifié + URI complet
  - **alias** : vers le chemin spécifié + URI sans le contexte
- Pour une redirection :
  - **return** : vers le chemin ou l'URL spécifiée avec un code

```
location /foo {
    root /var/www;
    alias /var/www;
    return 301 /foo/bar;
}
```

NGINX

Authentification basique

- Assez simple à mettre en place, très similaire à Apache
- On doit utiliser l'utilitaire **htpasswd** également pour le fichier de mots de passe
- Pas de possibilité de spécifier seulement certains utilisateurs comme sur Apache
- On peut également bloquer l'accès par rapport à une IP

```
location / {
    auth_basic "Administrator's Area";
    auth_basic_user_file /etc/nginx/.htpasswd;
    deny 192.168.1.1/24;
}
```

NGINX

Reverse Proxy

- Une des spécialités de Nginx simple à mettre en place
- Différents types de proxy à disposition : **HTTP**, **FastCGI**, etc.
- On utilise aussi des paramètres pour transmettre également les headers orignaux par exemple
- Il est aussi possible de référencer un groupe de serveur

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