Introduction au Serveur Web Nginx

Djelloul Ziadi

9 octobre 2024

Qu'est-ce que Nginx ?

- Nginx est un serveur web open-source, développé en `2004` par `Igor Sysoev`.
- Utilisé comme serveur HTTP, proxy inverse, et équilibreur de charge.
- Répond à la problématique de gestion des connexions simultanées (C10K : le problème de l'optimisation des sockets réseau pour gérer un grand nombre de clients en même temps).
- Basé sur une architecture événementielle asynchrone et non bloquante (`select`, `poll`, `kqueue` **epoll**).

Fonctionnalités principales de Nginx

- **Serveur HTTP** : Gestion des fichiers statiques (HTML, CSS, etc.).
- **Proxy Inverse** : Proxy inverse pour relayer les requêtes vers un autre serveur “backend".
- **Proxy de messagerie** : Supporte IMAP, POP3 et SMTP.
- **Serveur TLS/SSL** : Sécurisation des connexions avec HTTPS.
- **Equilibreur de Charge** : Répartition de la charge entre plusieurs serveurs.

Quelques Exemples de Configurations

Exemple 1 : Serveur HTTP

```
server {
 listen 80;
 server_name ssi.edu www.ssi.edu;

 root /var/www/ssi;
 index index.html;

 access_log /var/log/nginx/exemple_access.log;
 error_log /var/log/nginx/exemple_error.log;

 location / {
   try_files $uri $uri/ =404;
 }
}
```

Exemple 2 : Proxy Inverse

```
server {
  listen 80;
  server_name api.ssi.edu;

  location / {
    proxy_pass http://127.0.0.1:8080;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
  }
}
```

Exemple 3 : Equilibreur de Charges

```
upstream dorsale {
  server back1.ssi.edu;
  server back2.ssi.edu;
  server back3.ssi.edu;
}

server {
  listen 80;
  server_name www.ssi.edu;

  location / {
    proxy_pass http://dorsale;
  }
}
```

Exemple 4 : Serveur TLS/SSL

```
server {
  listen 443 ssl;
  server_name www.ssi.edu;

  ssl_certificate /etc/nginx/ssl/ssi.crt;
  ssl_certificate_key /etc/nginx/ssl/ssi.key;

  ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
  ssl_ciphers HIGH:!aNULL:!MD5;

  location / {
    root /var/www/ssi;
    index index.html;
  }
}

server {
  listen 80;
  server_name www.ssi.edu;
  return 301 https://$host$request_uri;
}
```

Fichier de Configuration **nginx.conf**

Structure du fichier `nginx.conf`

Le fichier de configuration `nginx.conf` utilise deux notions :

- **Contextes** : définissent le cadre dans lequel des directives sont appliquées.
- **Directives** : commandes spécifiques qui agissent sur le comportement de Nginx.

Les Contextes Principaux

- `"principale"` : définit les paramètres globaux.
- `http` : configure les paramètres relatives aux requêtes HTTP.
- `server` : définit la configuration d'un serveur virtuel.
- `location` : configure le comportement pour une route.
- `events` : permet de configurer la gestion des connexions entrantes et sortantes.
- `upstream` : permet définir un groupe de serveurs backend.

Les Types de Directives

- Simple

```
listen 80;
```

- Tableau

```
acccess_log /var/log/nginx/access.log
acccess_log /var/log/nginx/customlog.gz custom_format;
```

- Action

```
return 403 "Access Interdit !"
```

Exemple de fichier nginx.conf

```
acccess_log /var/log/nginx/access.log

http {
  index index.php index.html;
    server {
        listen 80;
        server_name ssi.edu;

        location / {
            root /var/www/ssi;
            acccess_log /var/log/nginx/access_ssi.log;
        }
    }
    server {
        listen 80;
        server_name gil.edu;

        location / {
           root /var/www/gil;
        }
    }
}
```

Structure de base d'un bloc `server`

Un bloc `server` :

- Encapsulé dans un bloc `http`.
- Dédié à la configuration d'un serveur virtuel.

```
http {
  server {
    listen 80;
    server_name ssi.edu www.ssi.edu;
    location / {
      root /var/www/html;
      index index.html;
    }
  }
}
```

Directives principales du bloc `server`

Le bloc `server` contient des directives qui contrôlent les aspects suivants :

- **listen** : le port et l'interface d'écoute.
- **server_name** : le ou les noms de domaine associés au serveur.
- **root** : le répertoire racine du serveur.
- **index** : les fichiers d'index à servir par défaut.

Directive `listen`

La directive de type tableau `listen` détermine sur quel port et éventuellement sur quelle adresse réseau le serveur Nginx doit écouter.

```
server {
  listen 80;
  listen [::]:80;
}
```

- Port par défaut : 80 pour HTTP, 443 pour HTTPS.

Directive `server_name`

La directive `server_name` associe des noms de domaine ou adresse IP à un serveur spécifique.

```
server {
  listen 80;
  server_name ssi.edu www.ssi.edu 127.0.0.1;
}
```

- `*` : caractères génériques, par exemple.

```
server {
  listen 80;
  server_name *.ssi.edu;
}
```

Directives `root` et `index`

Les directives `root` et `index` définissent où Nginx doit chercher les fichiers à servir et quels fichiers index afficher par défaut.

```
server {
  root /var/www/html;
  index index.html index.htm;
}
```

- `root` : le répertoire racine pour servir les fichiers.
- `index` : le fichier à utiliser par défaut lorsque l'URL pointe vers un répertoire.

```
curl http://www.ssi.edu/
<=> curl http://www.ssi.edu/index.html
```

Bloc `location`

Le bloc `location` dans le bloc `server` permet de spécifier comment Nginx doit traiter les requêtes HTTP en fonction de l'URI.

Deux fonctionnalités :

- Routage des requêtes.
- Configuration de la réponse
- **Syntaxe** :
  `location [modificateur] location_match {}`.

Modificateur	Type de correspondance
=	exacte
^~	par préfixe le plus long sans expression régulière
~	par expression régulière sensible à la casse
~*	par expression régulière insensible à la casse
	par préfixe
@nom	par expression régulière insensible à la casse

Exemple de bloc `location`

```
server {
  location = /master {
    return 200 "Corespondance exacte";
  }

  location ^~ /master2 {
    return 200 "Corespondance exacte";
  }

  location ~ /master[0-9] {
    return 200 "Corespondance ER sensible";
  }

  location ~* /master[0-9] {
    return 200 "Corespondance ER insensibe";
  }

  location /master {
    return 200 "Corespondance par prefixe";
  }
}
```

Cas d'utilisation de `location`

- Serveur de fichiers statiques : toute requête commençant par `/videos` sera servie à partir du répertoire `/var/www/videos/`

```
server {
  location /videos  {
    root /var/www;
  }
}
```

- Redirection : redirection permanente (code 301) d'`ancien` vers `nouveau`.

```
server {
  location /ancien/ {
    return 301 /nouveau/;
  }
}
```

Cas d'utilisation de `location`

- Proxy inverse.

```
server {
  location /api/ {
    proxy_pass http://backend;
  }
}
```

- Gestion des erreurs

```
server {
  error_page 500 502 503 504 /50x.html;
  location = /50x.html {
    root /var/www/erreurs;
  }
}
```

Cas d'utilisation de `location`

- Accès par authentification.

```
server {
  location /admin/ {
    auth_basic "Zone protegee";
    auth_basic_user_file /etc/nginx/.htpasswd;
  }
}
```

- Compression des données.

```
server {
  location ~* \.(css|js|html|svg)$ {
    gzip on;
    gzip_types text/css application/javascript text/html
        image/svg+xml;
  }
}
```

Cas d'utilisation de `location`

- Filtage par adresses IP.

```
server {
  location /secure/ {
    allow 192.168.1.0/24;
    deny all;
  }
}
```

- Filtrage par adresses IP ou mot de passe.

```
server {
  location /notes/ {
    satisfy any;

    allow 192.168.1.0/24;
    deny all;

    auth_basic "Zone protegee";
    auth_basic_user_file /etc/nginx/.htpasswd;
  }
}
```

Les variables

- Variables prédéfinies (ex. `$host`, `$uri`, `$args`, `$http_user_agent`).
  voir `https://nginx.org/en/docs/varindex.html`
- Variables utilisateur personnalisées : `set $nom = valeur`
  (ex. `set $mobile = 1`)

```
server {
  server_name ssi.edu;

  if ($http_user_agent ~* "Android|iPhone") {
    set $mobile 1;
  }

  if ($mobile) {
    rewrite ^ http://m.ssi.edu$request_uri? permanent;
  }
}
```

La directive d'action `rewrite`

La directive d'action `rewrite` permet de réécrire les URL des requêtes entrantes avant de les traiter. Elle utilise les expressions rationnelles

- Syntaxe : `rewrite regex replacement [drapeau];`
  - `regex` : L'expression régulière qui correspond à l'URL.
  - `replacement` : La nouvelle URL après réécriture.
  - `drapeau` : (optionnel) `last`, `break`, `redirect`, ou `permanent`.

```
rewrite ^/ancien/(.*)$ /nouveau/$1 permanent;
```

```
rewrite ^/ancien/(.*)$ /nouveau/$1 redirect;
```

```
rewrite ^/td/([0-9]+)$ /td.php?num=$1 last;
```

```
rewrite ^/td/([0-9]+)$ /td.php?num=$1 break;
```

Exemples de la directive `rewrite`

```
server {

  rewrite ^/formation/(\w+) /master/$1 last;
  rewrite ^/master/ssi /ssi.png;

  location /master {
    return 200 "Salut Master";
  }

  location = /master/ssi {
    return 200 "Salut Master SSI";
  }
}
```

Question : `http://localhost/formation/ssi` ?

La directive `try_files`

- `try_files` : permet de vérifier l'existence de fichiers ou de répertoires et et servir le premier qui existe
- **Syntaxe** : `try_files path1 [path2] fin`

```
server {

  location / {
    root /var/www/html;
    try_files $uri $uri/ $uri.html /index.html;
  }
}
```

Si la requête est `http://www.ssi.edu/toto` Teste l'existence de `/var/www/html/toto`, s'il n'existe pas teste `/var/www/html/toto/`, s'il n'existe pas teste `/var/www/html/toto.html`, s'il n'existe pas servir `http://www.ssi.edu/index.html`

La directive try_files

```
server {

  location / {
    root /var/www/html;
    try_files $uri $uri/ $uri.html =404;
  }
}
```

Si la requête est http://www.ssi.edu/toto Teste l'existence de /var/www/html/toto, s'il n'existe pas teste /var/www/html/toto/, s'il n'existe pas teste /var/www/html/toto.html, s'il n'existe pas retourner le code 404

Exemple try_files

```
events {}

http {
  include mime.types;
  server {
    listen 80;
    server_name *.ssi.edu;
    root /usr/share/nginx/html;

    try_files $uri $uri.png /master @amicalement_404;

    location @amicalement_404 {
      return 404 "Ooof, fichier introuvable !";
    }

    location /master {
      return 200 "Bienvenue a la formation master";
    }
  }
}
```

Journalisation

Deux types de journaux :

- Journaux d'accès (`access_log`) : Enregistre toutes les requêtes reçues par le serveur.
- Journaux d'erreur (`error_log`) : Enregistre les erreurs rencontrées par le serveur.
- `https://nginx.org/en/docs/http/ngx_http_log_module.html`

```
http {
  access_log /var/log/nginx/access.log;
  error_log /var/log/nginx/error.log;
}
```

Format personnalisé

```
http {
  log_format ssi '$remote_addr - "$request_uri"';
  access_log /var/log/nginx/access_ssi.log ssi;
}
```

Performance & Sécurité

Voici quelques points permettant d'optimiser les performances de votre serveur nginx :

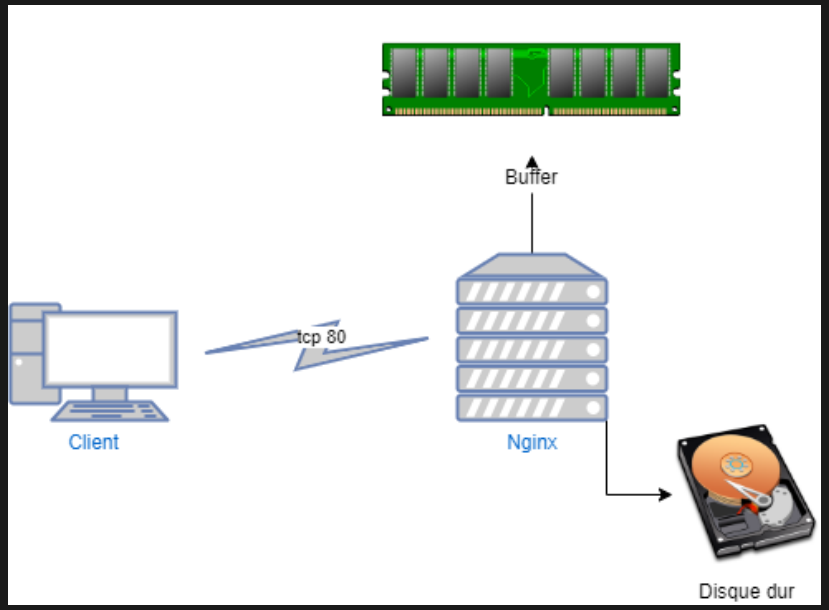
1. Nombre de `worker`
2. Tailles des buffers E/S
3. Paramétrage des `timers` (timeout)
4. Mise en cache
5. Compression
6. Chiffrement
7. HTTP2
8. HTTP/2 Server push
9. Compilation de nginx avec les modules nécessaires
10. Sécurisation du serveur ...

1. Nombre de workers

- La directive `worker_processes` définit le nombre de processus *worker*.
- Idéalement, il doit être égal au nombre de cœurs CPU (`nproc`, `lscpu`).
- La valeur `auto` permet à Nginx de déterminer automatiquement le nombre de cœurs disponibles.
- Exemple :

```
worker_processes auto;
```

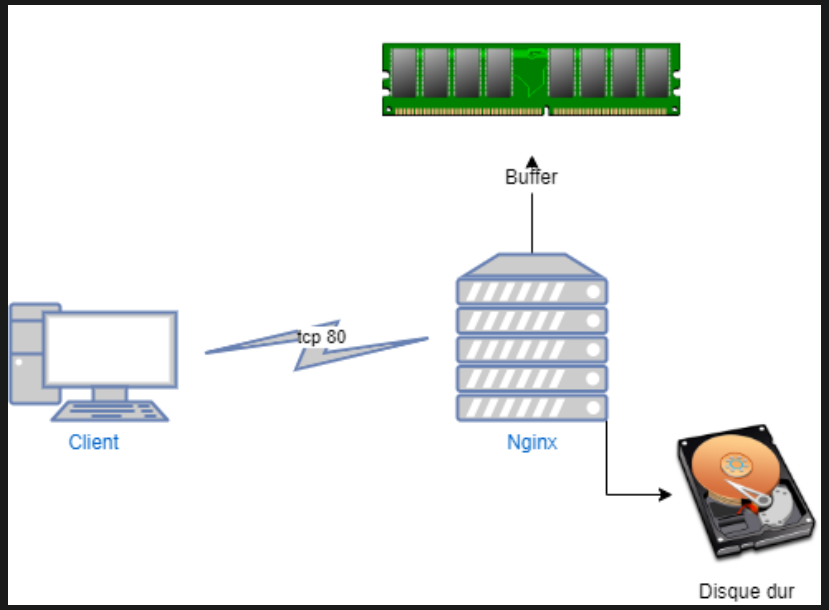
2. Taille des buffers E/S



- Ajuster la taille des buffers d'entrée et de sortie permet de mieux gérer le flux des données.
- Exemple

```
# Taille du Buffer pour les envois via POST
client_body_buffer_size 16K;
# Taille du Buffer pour les entetes
client_header_buffer_size 1K;
...
```

2. Taille des buffers E/S (2/2)

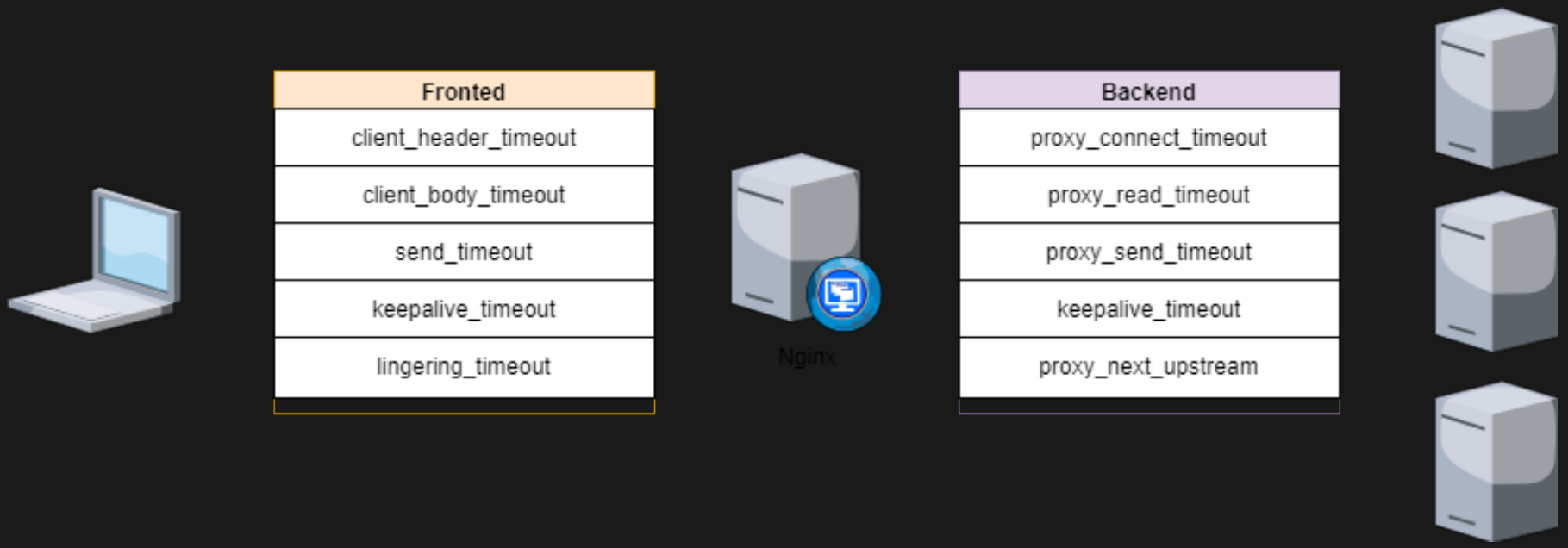


- `sendfile on;` permet d'ignorer la mise en mémoire tampon pour l'envoi du contenu statique.
- voir la fonction `sendfile()`.

```
# Ignorer la mise en memoire tampon pour les fichiers statiques
sendfile on;
```

Paramétrage des `timers` (timeout)

- Les paramètres de timeout contrôlent combien de temps Nginx attend une connexion ou une requête avant de la fermer.
- Il est crucial d'optimiser afin d'éviter la surcharger le serveur.



Frontend timeout

Client Header Timeout

- **Directive** : `client_header_timeout`
- **Valeur par défaut** : 60s
- **Description** : Définit un timeout pour la lecture de l'en-tête de la requête client. Si l'en-tête n'est pas transmis en entier dans ce délai, la requête est terminée avec une erreur 408 (Request Time-out).
- **Utilisation** : Prévention contre des attaques DDoS de type Slowloris.

Frontend timeout

Client Body Timeout

- **Directive** : `client_body_timeout`
- **Valeur par défaut** : 60s
- **Description** : Définit un timeout pour la lecture du corps de la requête client. Le timeout est mesuré entre deux opérations de lecture successives. Si rien n'est transmis dans ce délai, la requête est également terminée avec une erreur 408.
- **Utilisation** : Prévention des attaques DDoS de type Slowloris.

Frontend timeout

Send Timeout

- **Directive** : `send_timeout`
- **Valeur par défaut** : 60s
- **Description** : Définit un timeout pour la transmission d'une réponse au client. Ce délai est mesuré entre deux écritures successives. Si le client ne reçoit rien dans ce temps, la connexion est fermée.
- **Utilisation** : Gestion des problèmes liés aux mauvaises requêtes. Par exemple, sur une connexion internet lente l'envoi de la réponse (ex. un fichier volumineux) prenne beaucoup de temps.

Frontend timeout

Keep Alive Timeout

- **Directive** : `keepalive_timeout`
- **Valeur par défaut** : 75s
- **Description** : Définit un timeout pendant lequel une connexion keep-alive reste ouverte côté serveur. Le premier paramètre détermine la durée d'ouverture de la connexion, tandis que le second peut définir la valeur dans l'en-tête de réponse "Keep-Alive : timeout=time".
- **Utilisation** : Optimisation des performances pour les connexions multiples.

Frontend timeout

Lingering Timeout

- **Directive** : `lingering_timeout`
- **Valeur par défaut** : 30s
- **Description** : Définit le temps maximal d'attente pour des données supplémentaires du client après que le serveur a décidé de fermer la connexion, lorsque lingering_close est activé.
- **Utilisation** : Amélioration de la gestion des connexions inactives.

Backend timeout (1/2)

- `proxy_connect_timeout` – Définit le temps maximum pour se connecter à un serveur proxy "backend" (60 par défaut, généralement moins de 75)
- `proxy_read_timeout` – Définit le temps maximum (entre deux lectures) pour lire la réponse du serveur proxy (60 par défaut)
- `proxy_send_timeout` – Définit le temps maximum (entre deux écritures) pour envoyer la requête du serveur proxy (60 par défaut)

Backend timeout (2/2)

- `resolver_timeout` - Définit un timeout pour la résolution de noms des backend, spécifiant combien de temps Nginx attendra une réponse du résolveur (DNS).
- `proxy_next_upstream` - directive définit dans quels cas Nginx doit essayer un autre serveur backend
- `proxy_next_upstream_timeout` Limite le temps pendant lequel une requête peut être transmise au serveur suivant. La valeur 0 désactive cette limitation.

Mise en cache

Introduction à la mise en cache

- Permet d'accélérer les temps de réponse en réduisant les appels aux serveurs backend.
- **Deux types de mise en cache** :
  - **Cache statique** : Pour les fichiers statiques (CSS, JS, images, etc.).
  - **Cache dynamique** : Pour les réponses générées dynamiquement (HTML, JSON, etc.).

Cache statique

- Utilisé pour stocker des fichiers statiques.
- Réduit le besoin d'accéder au disque ou au serveur d'application.
- `expires` : Définit la durée de mise en cache des fichiers statiques.
- `access_log off` : Désactive la journalisation.
- **Exemple** :

```
location ~* .(jpg|jpeg|png|gif|ico|css|js)$ {
  expires 60d;
  access_log off;
}
```

Cache dynamique

- Utilisé pour stocker des réponses générées dynamiquement par un serveur backend.
- Réduit la charge sur les serveurs d'application.
- `proxy_cache` : Définit la zone de cache pour les réponses proxy.
- `proxy_cache_valid` : Spécifie la durée de validité des réponses en cache.

```
http {
  proxy_cache_path /var/cache levels=1:2 keys_zone=moncache:15m;
  location / {
    proxy_pass http://backend;
    proxy_cache moncache;
    proxy_cache_valid 200 302 10m;
    # code reponse 200 ou 302
    # duree dans le cache 10m
  }
}
```

Compression

Introduction à la compression

- Permet de réduire la taille des réponses HTTP.
  - Diminution de l'utilisation de la bande passante.
  - Amélioration de la vitesse de chargement des pages.
- Pré-compression : `gzip_static` ou `brotli_static` ;
- Compression à la volée : `gzip` ou `Brotli`

Compression Gzip (1/2)

- **Gzip** est l'algorithme de compression le plus couramment utilisé avec Nginx.
- Réduit la taille des fichiers envoyés au client (HTML, CSS, JS, etc.).
- `gzip` : Active ou désactive la compression.
- `gzip_types` : Spécifie les types MIME à compresser.
- `gzip_min_length` : Définit la taille minimale des réponses compressées.
- `gzip_compression_level` : Définit le niveau de compression.
- **Exemple** :

```
gzip on;
gzip_types text/plain text/css application/javascript;
gzip_min_length 1000;
gzip_comp_level 4;
```

Compression Gzip (2/2)

Types de contenu compressibles

- types de contenu compressibles :
  - Textes (HTML, CSS, JavaScript, etc.)
  - JSON, XML
- A ne pas compresser certains types comme :
  - Images (JPEG, PNG, etc.) : déjà compressées.
  - Vidéos et fichiers binaires.

Niveaux de compression

- Le niveau de compression Gzip est compris entre 1 et 9.
- `Niveau 1` : Compression rapide mais avec un faible taux.
- `Niveau 9` : Compression maximale mais plus lente.
- `Bon compromis` : Utiliser un niveau entre 4 et 6.

Pré-compression

Compression des fichiers statiques :

```
find /var/www/static/ -type f -mmin -15 -regextype
    posix-extended -iregex '.*\.(css|html|js|json|txt|xml)'
    -exec gzip -k {} \;
```

Activation de la pré-compression gzip

```
server {
  listen 80;
  server_name ssi.edu;

  location / {
    root /var/www/static;
    gzip_static on;
  }
}
```

Chiffrement

Introduction à TLS

- `TLS (Transport Layer Security)` : Protocole pour sécuriser les communications sur Internet.
- Permet de garantir la confidentialité et l'intégrité de l'échange de données.
- `Nginx` supporte `TLS` pour établir des connexions HTTPS.

Configuration de base pour HTTPS

Pour activer TLS, avec un certificat SSL/TLS.

- Active le port HTTPS : `listen 443 ssl;`
- Définir le chemin vers votre certificat SSL : `ssl_certificate`
- Définir le chemin vers votre clé privée : `ssl_certificate_key`
- **Exemple de configuration** :

```
server {
  listen 443 ssl;
  server_name www.ssi.edu;

  ssl_certificate /etc/nginx/ssl/ssi.crt;
  ssl_certificate_key /etc/nginx/ssl/ssi.key;
}
```

Configuration sécurisée de TLS

- Utilisez des paramètres de sécurité pour renforcer la configuration TLS.
- Utiliser uniquement les versions sûres de TLS : `ssl_protocols TLSv1.2 TLSv1.3;`.
- Définir les algorithmes de chiffrement autorisées : `ssl_ciphers`
- Pour une configuration plus complète consulter le site `https://ssl-config.mozilla.org/`
- Pour tester la sécurité de votre site `https://www.ssllabs.com/ssltest/`

Redirection HTTP vers HTTPS

- `return 301 https://$host$request_uri;`
- **Exemple** :

```
server {
  listen 80;
  server_name www.ssi.edu;
  return 301 https://$host$request_uri;
}
```

Configurer la sécurité de transport strict

- HSTS : HTTP Strict Transport Security
- IETF dans la RFC 6797 en 2012.
- Forcer le navigateur à utiliser des connexions sécurisées (HTTPS).
- `add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;`
- **Exemple** :

```
add_header Strict-Transport-Security "max-age=31536000;
    includeSubDomains" always;
```

- Tester en utilisant un scanner `https://www.ssllabs.com/ssltest/`

HTTP/2 ?

- Successeur de HTTP/1.1
- Publié en 2015 par l'IETF
- Protocole en binaire
- Permet d'améliorer les performances (vitesse de chargement des pages)
- Permet une meilleure utilisation de la bande passante
- Renforce la sécurité avec TLS (nécessite TLS)
- Adoption croissante par la majorité des navigateurs et des serveurs web

Caractéristiques

Afin d'améliorer les performances HTTP/2 utilise :

- Multiplexage des requêtes (ex. HTML+CSS+JS regroupé dans un seul fichier binaire)
- Compression des en-têtes
- Utilisation de connexions persistantes
- Server push (ex. req = index.html => rep = (index.html, style.css, script.js))

Exemple de configuration

```
server {

  listen 443 ssl http2;
  server_name www.ssi.edu;

  root /var/www/ssi;

  index index.php index.html;

  ssl_certificate /etc/nginx/ssl/ssi.crt;
  ssl_certificate_key /etc/nginx/ssl/ssi.key;

  location / {
    try_files $uri $uri/ =404;
  }
}
```

- Pour une configuration plus complète consulter le site `https://ssl-config.mozilla.org/`
- Pour tester la sécurité de votre site `https://www.ssllabs.com/ssltest/`

Merci !