nginx

Configuration

- worker_processes : par défaut debian met `auto`, donc le nombre de processeurs alloués à la VM
- worker_connections : par défaut debian met 768

NB : contrairement à apache, en cas de dépassement

- prévient par un "worker_connections are not enough" dans error.log
- renvoie une réponse vide au client
- aucune entrée dans le access log !!

proxy_pass

Préférer

```
proxy_pass http://xxx
```

sans `/` à la fin. Cela indique à nginx de conserver le même path. Cela fonctionne notamment dans un `location` avec regexp (~)

nested location

Il est possible d'avoir des `location` nested, mais le `proxy_pass` n'est pas hérité, il faut donc le répéter. Si ajout d'un `add_header`, ceux des niveaux supérieures sont ignorés.

Il est possible d'utiliser `if` à la place, notamment pour faire des `add_header`. nginx vous dira si une directive est autorisé dans un `if`. C'est aussi indiqué dans la documentation.

compression

Par défaut seul le contenu text/html est compressé. Il est conseillé d'ajouter des gzip_types comme fait par défaut sur les apache sous debian.

Il faut aussi s'assurer que `gzip on` est activé (c'est le cas sur debian, mais pas sur CentOS)

headers avec des "_"

Par défaut, les headers avec des "_" sont bloqués. Pour les autoriser, utiliser underscores_in_headers.

(exemple : nécessaire si l'on veut autoriser "REMOTE_USER")

WebSocket

Il faut s'assurer de passer les headers "Upgrade" et "Connection" de la 1ère requête WebSocket (le handshake).

Nginx va ensuite détecter la réponse d'Upgrade vers WebSocket et établir un tunnel.

Tout est clairement documenté upstream : https://nginx.org/en/docs/http/websocket.html

Monitoring

Par défaut le monitoring nginx est très limité : module stub_status. Ce module est utilisé par /usr/local/bin/monitor-nginx .

Pour avoir les “live requests”, on peut ajouter du code lua qui le fait, cf monitor-live-requests.conf (à mettre dans /etc/nginx/conf.d/ par exemple)

Attention : cela nécessite un nginx >= 1.11 (donc debian 10 ou debian 9 avec backports)

Différences avec Apache

Limitation de la taille d'upload

Par défaut Apache limite très peu la taille d'upload, laissant le soin aux scripts de limiter.

Nginx en reverse proxy limite par défaut l'upload à 1M. Il faut l'augmenter (surtout pour des sites permettant l'upload de fichiers) :

/etc/nginx/conf.d/max_body_size.conf

```
client_max_body_size 150M;
```

Cache

Attention à `proxy_cache_key` :

- contrairement à ce qu'indique la documentation, le cache ignore http vs https. D'où des pbs de mixed-content
- si le même backend est utilisé pour plusieurs vhosts, ils vont se mélanger.

Solution à ces deux pbs, forcer une valeur `proxy_cache_key` plus solide :

```
proxy_cache_key $scheme://$host$request_uri;
```

Pour faire fonctionner le force-reload :

```
proxy_cache_bypass $http_pragma;
```

Pour forcer une mise en cache d'un contenu, il faut comprendre ce qu'effectue nginx :

- interrogation du cache avec "proxy_cache_key", en vérifiant les contraintes sur les request headers (voir plus loin)
- si cache trouvé et si évaluation de "proxy_no_cache" répond faux
  - utilisation du cache
- sinon
  - requesting upstream après effectué "proxy_set_header"
  - si réponse n'a pas "Set-Cookie" ou si ignoré par "proxy_ignore_headers"
    et si réponse a X-Accel-Expires/Cache-Control/Expires et non ignoré par "proxy_ignore_headers"
    ou si "proxy_cache_valid" l'autorise
    - si response headers contient "Vary" et non ignoré par "proxy_ignore_headers"
      - si "Vary: *" ou Vary trop long (plus de 128 chars)
        - pas de mise en cache
      - sinon mise en cache avec contrainte sur les request headers listé dans "Vary"
    - sinon mise en cache sans contrainte
  - sinon pas de mise en cache
- dans tous les cas, effectuer "proxy_hide_header" et "add_header"

PS : la durée de mise en cache est définie par X-Accel-Expires/Cache-Control/Expires/proxy_cache_valid (avec cet ordre de priorité)

Buffering

Par défaut nginx bufferise les requêtes et les réponses. Cela peut nécessiter de la place disque.

Pour voir ces fichiers temporaires , on peut utiliser :

```
find /proc/[0-9]*/fd -lname '*(deleted)' 2>/dev/null | perl -lne 'print -s, " $_ $l", readlink'  | sort -g
```

Buffering des réponses

Bufferiser les réponses est utile quand le client est lent : le backend donne la réponse entière et peut passer à autre chose (utile notamment pour des backend apache-prefork ou PHP-FPM gourmand en mémoire).

Si la réponse est trop grosse pour être bufferisée en mémoire la réponse est stockée dans un fichier temporaire. Son emplacement est défini à la compilation sur debian

```
% nginx -V 2>&1 | perl -lne 'print $& while /--(\S+)/g' | grep proxy-temp-path
--http-proxy-temp-path=/var/lib/nginx/proxy
```

Si la réponse est vraiment grosse, nginx ne bufferise que le début. Par défaut nginx bufferise les réponses jusqu'à 1G. Il est possible de réduire la taille via proxy_max_temp_file_size.

Il est aussi possible de désactiver le buffering complètement avec `proxy_buffering off` mais attention, cela désactive aussi la fonctionnalité `proxy_cache`

Gros fichiers et timeout backend

Le frontal nginx va bufferiser d'un coup `proxy_max_temp_file_size` puis demander le reste au fur et a mesure. Réf

Il faut que le timeout du backend soit supérieur au temps que met le client à télécharger ces premiers `proxy_max_temp_file_size`, sinon le backend va abandonner et on verra l'erreur `upstream prematurely closed connection while reading upstream` dans le error log de nginx.

Par exemple, si on veut autoriser un téléchargement en modem 56kbps et que `proxy_max_temp_file_size` est 30MB, il faut un Timeout / send_timeou backend de 2h.

Upload buffering

Si la requête est plus grosse que client_body_buffer_size la requête est stockée dans un fichier temporaire. Son emplacement est défini à la compilation sur debian

```
% nginx -V 2>&1 | perl -lne 'print $& while /--(\S+)/g' | grep client-body-temp-path
--http-client-body-temp-path=/var/lib/nginx/body
```

Si nginx est utilisé en reverse proxy sans load balancing, on peut désactiver cette fonctionnalité :

```
proxy_request_buffering off;
proxy_http_version 1.1;
```

Content-Type par défaut

Apache accepte de ne pas renvoyer de `Content-Type`. Nginx va forcer un Content-Type par défaut, souvent `application/octet-stream` ce qui va indiquer au navigateur de sauvegarder un fichier plutôt que d'afficher une page Web par exemple.

backends et IPv4/IPv6

- apache : il semble qu'il préfère l'IPv6 du serveur backend (cf `apr_sockaddr_info_get` qui utilise getaddrinfo et donc RFC 3484 puis `ap_proxy_connect_backend` qui va prendre le premier qui fonctionne)
- nginx : utilise IPv4 ou IPv6 en alternance (cf `ngx_http_upstream_init_round_robin` qui utilise `ngx_inet_resolve_host`)

Certaines applications (comme typo3) n'apprécient pas ce changement d'IP. Solution pour typo3 en backend apache+mod-php : configurer remote_ip… en espérant que le client ne mixe pas les requêtes IPv4 et IPv6 (cf Happy eyeballs) !!

Un autre problème avec l'alternance IPv4/IPv6 : nginx va considérer les deux IPs comme des backends différents, et va donc gérer le status de ces backends séparément. Cela peut provoquer des erreurs "no live upstreams". Solutions possibles : remplacer le nom par l'IPv4 dans la directive ProxyPass, ou supprimer l'entrée DNS IPv6 du backend.

TLS session tickets

Par défaut, nginx n'a pas de cache de TLS session tickets. Il faut l'activer avec `ssl_session_cache`.

POST on static pages

nginx autorise uniquement la méthode GET sur les pages statiques.

Solution possible : `"error_page 405 =200 $uri;"`

Techniques de proxification

Modifier un response header

- Pour forcer une valeur :

proxy_hide_header Content-Security-Policy;
add_header Content-Security-Policy "xxx";

- Pour ajouter une valeur

proxy_hide_header Content-Security-Policy;
add_header Content-Security-Policy "$upstream_http_content_security_policy; xxx";

- Pour modifier une valeur

```
# NB: doit être fait en scope global ("http")
map $upstream_http_content_security_policy $content_security_policy_allow_bandeau {
    '~(.*); script-src (.*)' "$1; script-src ent.univ.fr $2";
    '~(.*)' "$1; script-src ent.univ.fr";
}

proxy_hide_header Content-Security-Policy;
add_header Content-Security-Policy $content_security_policy_allow_bandeau;
```

Proxifier un backend avec SNI

```
proxy_pass https://foo.bar;
proxy_ssl_verify on;
proxy_ssl_verify_depth 1;
# le SNI est utilisé par le backend
proxy_ssl_name xxx.univ.fr;
proxy_ssl_server_name on;
```

Mais attention si on a besoin d'un deuxième SNI vers le même serveur backend, exemple :

```
proxy_pass https://foo.bar;
proxy_ssl_name xxx2.univ.fr;
```

=> cela ne fonctionnera pas à cause "**`proxy_ssl_session_reuse`**".

Une solution est "**`proxy_ssl_session_reuse off`**".

Une autre solution est d'utilisateur un upstream :

```
# NB: pour proxy_pass https://xxx.univ.fr, il semble que nginx utilise "xxx.univ.fr" comme clé pour la
fonctionnalité proxy_ssl_session_reuse
# En utilisant un upstream, on peut forcer une clé différente pour un même backend
upstream upstream_xxx2 {
    server foo.bar:443;
}
server {
   ...
   proxy_pass https://upstream_xxx2;
   proxy_ssl_name xxx2.univ.fr;
}
```

Sticky sessions avec nginx open source

```
upstream prefer_server1 {
    server server1:8080;
    server server2:8080 backup;
}

upstream prefer_server2 {
    server server1:8080 backup;
    server server2:8080;
}

...
    set $prefered_backend server2;
    if ($cookie_JSESSIONID ~ "[.](server1|server2)$") { set $prefered_backend $1; }
    proxy_pass http://prefer_$prefered_backend;
    include proxy_params;
```

Envoyer une requête à tous les backends

Si vous utilisez l'affinité de session pour choisir le backend, deux pbs peuvent se poser :

- CAS Single Logout back-channel : le serveur CAS fait une requête directement au serveur, sans passer par le navigateur. Il n'envoie donc pas de cookie de session qui permettrait de savoir quel backend choisir
- proxy CAS : le serveur CAS envoie le PGT directement à l'application, sans passer par le navigateur. Il est souvent possible de configurer l'application pour fournir à CAS une URL directe, ne passant pas par le balancer.

```
upstream prefer_server1 {
    server server1:8080;
    server server2:8080 backup;
}

upstream prefer_server2 {
    server server1:8080 backup;
    server server2:8080;
}

server {
    ...

    set $prefered_backend server3;
    set $fallback_backend server2;
    ...
        error_page 418 = @broadcast;
        set $need_broadcast 0;
        if ($remote_addr = 192.1.2.3) { set $need_broadcast 1; }
        if ($need_broadcast) { return 418; }
    ...

    location @broadcast {
        mirror /@mirror_to_fallback;
        mirror_request_body on; # for CAS back-channel Single-Logout
        proxy_pass http://${prefered_backend}.univ.fr:8080;
        include proxy_params;
    }
    location /@mirror_to_fallback {
        proxy_pass http://${fallback_backend}.univ.fr:8080$request_uri;
        include proxy_params;
        proxy_set_header User-Agent "Duplicated-$http_user_agent";
    }
}
```

Shibboleth-SP

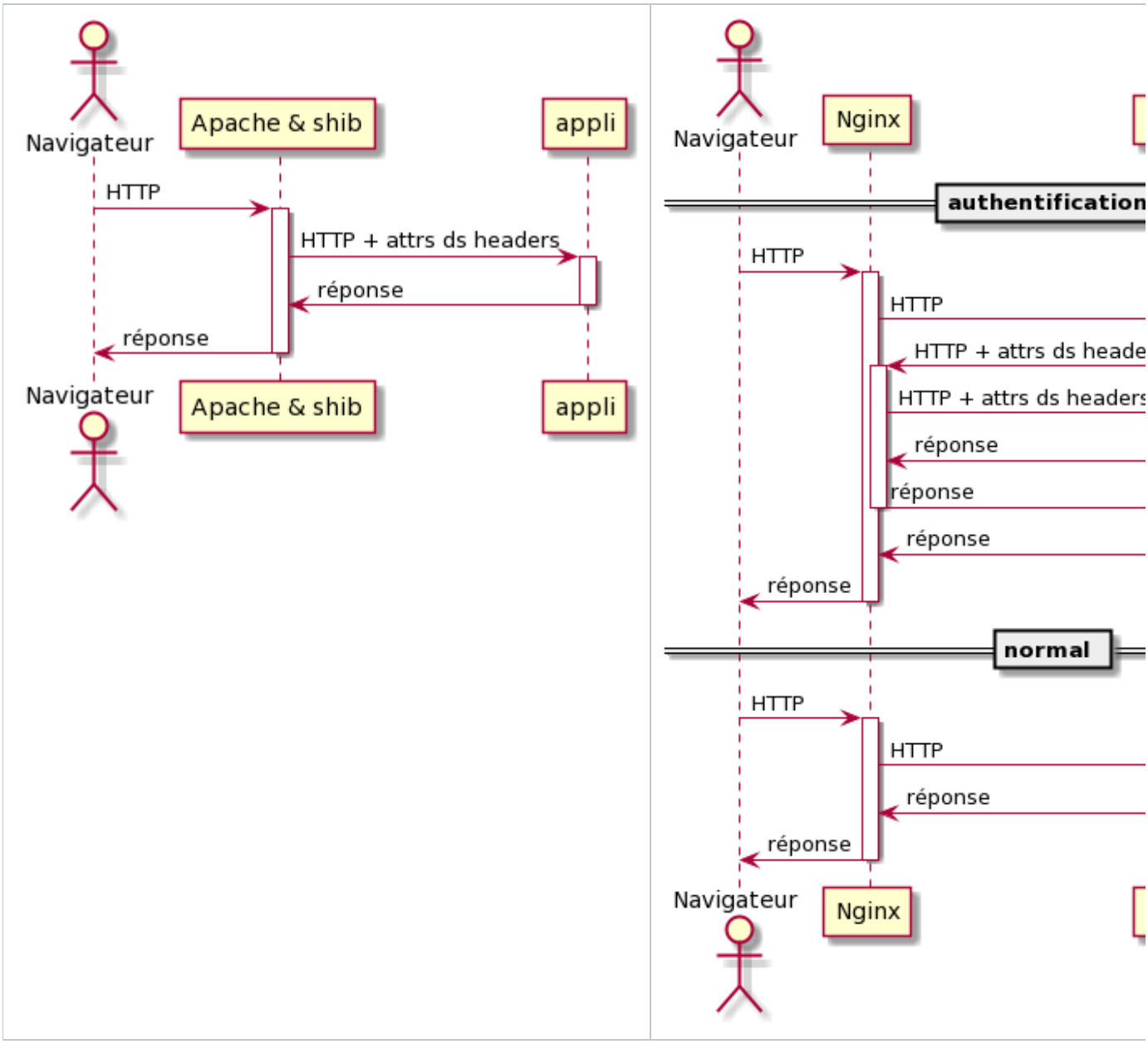
Shibboleth-SP fournit un FastCGI authorizer. Nginx ne gère pas cela en standard.

Le module nginx-http-shibboleth fournit cette fonctionnalité. Mais ce module n'est pas packagé en standard dans les distributions Linux.

Voici une solution alternative, testée sur moodle. NB : cette solution est coûteuse si l'application ne gère pas sa propre session (par exemple une appli qui protège toutes les urls avec shib).

Explication

Fonctionnement classique avec Apache et module shib2	Nginx renvoie les requêtes protégées par SAML vers Apache



(diagrammes de séquences réalisés avec plantuml et les fichiers apache-shib-simple.puml et nginx-shib-proxy.puml)

Configuration sur le nginx

```
set $shib_backend '172.0.0.42';

location /Shibboleth.sso {
    proxy_pass http://$shib_backend;
    proxy_set_header Host $http_host;
}

location ~ [^/]\.php(/|$) {
    set $uri_to_shib $uri;
    if ($remote_addr = $shib_backend) {
        set $uri_to_shib '';
    }
    if ($uri_to_shib = "/auth/shibboleth/index.php") {
        proxy_pass http://$shib_backend;
    }
    proxy_set_header Host $http_host; # preserve host when going to shib backend

    # standard FPM configuration
    fastcgi_split_path_info  ^(.+\.php)(/.+)$;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    fastcgi_param PATH_INFO $fastcgi_path_info;
    include fastcgi_params;

    fastcgi_pass unix:/run/php/moodle-fpm.sock;
    fastcgi_index index.php;
}
```

Configuration sur le apache/mod_shib

Configurer normalement shibboleth, et relayer les requêtes vers le nginx :

```
    ProxyPass / https://le_nginx/
    ProxyPreserveHost On
    ShibUseHeaders On
```