Reverse proxy jako mechanizm izolacji aplikacji webowych

Sebastian Kwaśniak Anna Berkowska Artur Binczyk Jerzy Szyjut
Damian Trowski

Wstęp teoretyczny

Po co izolować aplikacje webowe?

Izolacja aplikacji webowych jest kluczowa dla zapewnienia bezpieczeństwa, wydajności i niezawodności systemów. Dzięki izolacji, potencjalne zagrożenia i awarie w jednej aplikacji nie wpływają na działanie innych aplikacji w tym samym środowisku. Dodatkowo, izolacja ułatwia zarządzanie zasobami, pozwala na lepsze skalowanie oraz utrzymanie aplikacji, a także minimalizuje ryzyko konfliktów pomiędzy różnymi wersjami bibliotek i zależności. W konsekwencji, izolacja aplikacji webowych prowadzi do bardziej stabilnych i bezpiecznych systemów informatycznych.

Aktualnie każdy korzysta z izolacji pod różnymi postaciami, my skupimy się na tej oferowanej przez Reverse Proxy.

Czym jest reverse proxy?

Reverse proxy to serwer pośredniczący, który przyjmuje zapytania od klientów i przekazuje je do odpowiednich serwerów backendowych. W przeciwieństwie do tradycyjnego proxy, które działa w imieniu klienta, reverse proxy działa w imieniu serwera. Jego głównymi funkcjami są rozkładanie obciążenia (load balancing), zwiększanie wydajności dzięki cache'owaniu, zabezpieczanie aplikacji przed atakami DDoS oraz ukrywanie wewnętrznej struktury sieci. Reverse proxy może również terminować połączenia SSL, co upraszcza konfigurację certyfikatów na serwerach backendowych. W rezultacie, reverse proxy znacząco poprawia skalowalność, bezpieczeństwo i elastyczność infrastruktury sieciowej.

Czym to się różni od zwykłego proxy?

Reverse Proxy z definicji przekierowuje żądania od klientów zewnętrznych do serwerów wewnętrznych, a zwykłe proxy w drugą stronę, czyli od klientów wewnętrznych do zewnętrznych zasobów.

Ich cel użycia jest zupełnie inny dlatego nie mówimy tutaj o zwykłym proxy i pomijamy całkowicie ten temat.

Klient nie wie o istnieniu reverse proxy, widzi tylko jeden adres IP. Reverse Proxy może:

- Równoważyć obciążenie między wieloma serwerami (np. aplikacja potrafi być rozproszona pomiędzy wiele maszyn, to możemy połączenia raz dawać tu, raz tam na maszynę, raz na jeszcze inną)
- Terminować SSL - o tym już mówiliśmy
- Cache'ować zasoby - może szybciej podawać zasoby statyczne, takie jak obrazy lub proste rzeczy które są często pobieran e i się rzadko zmieniają.
- Ukrywać wewnętrzne struktury sieci - aplikacja sama nie będzie wiedziała o wewnętrznej sieci więc i jest struktura sieci jeszcze w dalszym zasięgu od użytkownika

Jak działa reverse proxy?

Można to sprowadzić do trzech prostych punktów:

1. Odebranie żądania przez Reverse Proxy
2. Odczytanie kluczowych parametrów do routowania:

- Nagłówki HTTP, w tym głównie "Host", w której jest domena do której chce połączyć się klient
- Dla połączenia SSL/TLS odczytanie rozszerzenia SNI (Server Name Identification)

- Metoda HTTP

3. Przekazanie żądanie do odpowiedniego serwera na podstawie parametrów

Wady reverse proxy

Reverse proxy, mimo licznych zalet, posiada również pewne wady:

Single Point of Failure (SPOF):

reverse proxy może stać się pojedynczym punktem awarii. Jeśli reverse proxy przestanie działać, dostęp do wszystkich usług backendowych może zostać zablokowany.

Nie jest to prawdziwa izolacja: Reverse proxy nie zapewnia pełnej izolacji pomiędzy serwerami backendowymi. W przypadku zagrożeń bezpieczeństwa lub awarii w warstwie proxy, mogą one wpłynąć na wszystkie obsługiwane aplikacje.

Dodaje opóźnienie: każde zapytanie musi przejść przez warstwę reverse proxy, co może wprowadzać dodatkowe opóźnienia, zwłaszcza w przypadku skomplikowanych operacji lub przeciążenia serwera proxy.

Większe koszta: wdrożenie i utrzymanie reverse proxy wiąże się z dodatkowymi kosztami, zarówno pod względem zasobów sprzętowych, jak i administracyjnych. Wymaga to dodatkowego zarządzania, monitorowania oraz potencjalnie bardziej złożonej infrastruktury sieciowej.

Jakie są rozwiązania reverse proxy

Mamy 4 główne wybory jeśli chodzi o Reverse Proxy.

NGINX: - Nginx robi wiele różnych rzeczy - Może forwardować traffic http i https, websocket, ostatnio nawet traffic TCP - Korzystany w Twitch.tv, Netflix - Wydany w 2004 roku, ciągle rozwijany - Ma dwie licencje: open source i komercyjną

HAProxy:

- Korzystany w Github, Reddit
- Rok wydania 2000, nadal aktualizowany
- Robi parę rzeczy i robi je bardzo dobrze
- Szukając informacji jest uznawany za najbardziej stabilne rozwiązanie

Pingora:

- Używany przez Cloudflare
- Pingora to rzecz nowa (rok wydania 2022), teoretycznie nieprzetestowana w boju (choć cloudflare twierdzi przeciwnie)
- Trochę cięższa do konfiguracji, bo to kod a nie pliki konfiguracyjne
- Cloudflare twierdziło że nginx jest za wolny i muszą "szybciej"

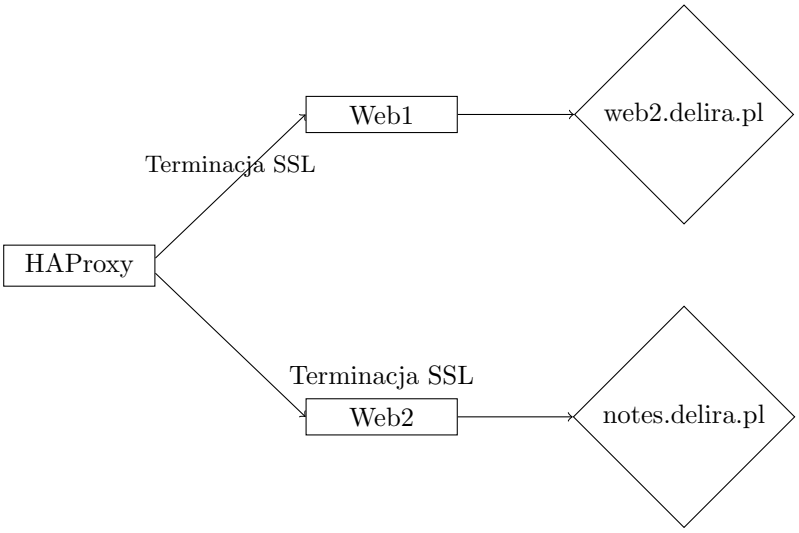
Traefik:

- Ani nie najszybsze, ani nie najbardziej stabilne
- Proste do wdrożenia w przypadku kontenerów (Docker, Kubernetes)

Nasz projekt

W naszym projekcie postawiliśmy dwie strony internetowe, pierwsza: `notes.delira.pl` oraz druga: `web2.delira.pl`. Obydwie są na osobnych maszynach. Przed nimi stoi jeden reverse proxy w postaci HAProxy. W przypadku `notes.delira.pl` certyfikat SSL terminowany jest na tej samej maszynie co aplikacja webowa, a w przypadku `web2.delira.pl` na maszynie z HAProxy.

Na schemacie wygląda to następująco:



Nginx

Nginx służy na obydwu maszynach (web1 oraz web2) jako serwer webowy. Konfiguracja dla `notes.delira.pl` wygląda następująco:

```
server {
        location ~ /\. {
            return 404;
        }
        listen 80;
        listen 443 ssl http2;
        server_name notes.delira.pl;
        ssl_dhparam  /etc/nginx/ssl/delira.pl/dhparams.pem;
        ssl_certificate /etc/nginx/ssl/delira.pl/delira.pl.fullchain.cer;
        ssl_certificate_key /etc/nginx/ssl/delira.pl/delira.pl.key;
        ssl_session_timeout 1d;
        ssl_session_cache shared:NixCraftSSL:10m;

        ssl_session_tickets off;
        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA38
        ssl_prefer_server_ciphers off;
        root /var/www/notes;

        index probale/zestaw-1..html;

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

Jak widzimy, nasłuchuje on na porcie 443 oraz ma wskazane gdzie znajduje się certyfikat SSL.

Konfiguracja `web2.delira.pl`:

```
server {
    server_name web2.delira.pl;
    listen 80;
    root /var/www/html;
    index index.html;
    location / {
        try_files $uri $uri/ =404;
    }
}
```

Jak widać jest o wiele prostsza, nasłuchuje na porcie 80 i nie ma żadnych informacji o jakimkolwiek szyfrowaniu czy certyfikatach SSL.

HAProxy

Tutaj sytuacja była bardziej skomplikowana i napotkaliśmy problem, ponieważ nie da się tak łatwo wykonać sytuacji opisanej na schemacie.

W naszym rozwiązaniu, chodzi o konkatenacją backendu z frontendem w HAProxy. Działa to w ten sposób, że domyślny backend w `main_ssl` łączy wszystkie zapytania, które nie będą przekazywane dalej, do `tcp_to_https`. Chodzi głównie o to, że jeżeli zapytanie nie jest do `notes.delira.pl` (lub możnaby tu dodać także zasady do innych domen, np. `web3.delira.pl`, `web4.delira.pl`, które miałyby własny certyfikat ssl na dalszej maszynie) to wyślij do domyślnego backendu: czyli siebie (`127.0.0.1`) na innym porcie, a wtedy frontend który nasłuchuje na porcie 8443 zajmuje się terminacją SSL.

```
frontend ssl_termination
        mode http
        bind *:8443 ssl crt /etc/haproxy/certs/web2.delira.pl.pem
        default_backend web2delira

frontend main_ssl
        bind :443
        mode tcp
        http-request set-src req.hdr(CF-Connecting-IP)

        use_backend delira_ssl if { req_ssl_sni -i  notes.delira.pl }

        default_backend tcp_to_https

backend tcp_to_https
        mode tcp
        server haproxy-https 127.0.0.1:8443 check

backend delira_ssl
        mode tcp
        balance roundrobin
        server delira_ssl_server 192.168.5.5:443

backend web2delira
        mode http
        server web2deliraserver 192.168.5.11:80 check
        http-request set-header X-Forwarded-Port %[dst_port]
        http-request add-header X-Forwarded-Proto https if { ssl_fc }
```

Podsumowanie

Podsumowując, reverse proxy to prosty sposób na lepszą izolację, lecz ma swoich parę wad, o których warto pamiętać. W większości przypadków jest to świetne rozwiązanie by mieć jedną maszynę, która jest stabilniejsza i nie posiada wielu zależności (np. Pythona, jakichś bibliotek itp.), aby wystarczająco uprościć zarządzanie maszyną i zmniejszyć możliwości ataku na nią.