

Ankieta

Daj nam feedback odnośnie prowadzonych zajęć i pomóż nam je ulepszyć uzupełniając tą ankietę: <https://forms.gle/y8opbHf7rNFZVfRx6>

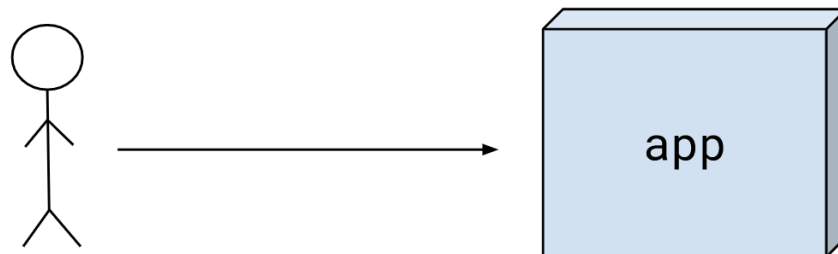
Agenda

1. Wprowadzenie do load balancingu
2. Load balancing DNS
3. **[Zadanie]** DNS resolving
4. Load balancing jako urządzenie fizyczne
5. **[Zadanie]** Load Balancer HTTP
6. Load balancing na różnych warstwach ISO/OSI
7. **[Zadanie]** Load balancer warstwy 4

Wiedza

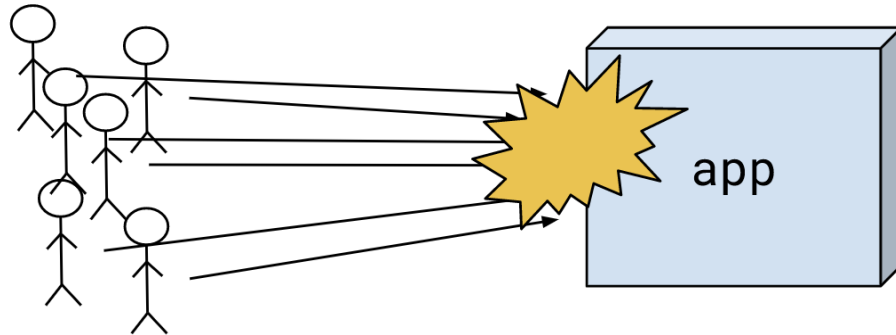
Dlaczego potrzebujemy load balancera?

Sytuacja



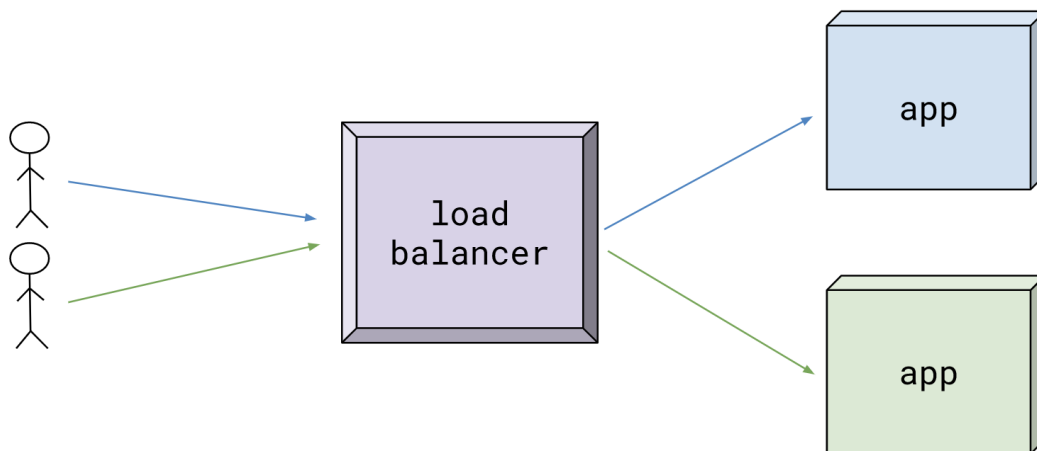
- Nasza aplikacja staje się popularna. Serwer dostaje coraz więcej ruchu i zużycie procesora rośnie.
- Analitycy przewidują, że liczba użytkowników naszej aplikacji podwoi się w najbliższym czasie.
- Musimy wdrożyć nową wersję aplikacji w której dodajemy krytyczną funkcjonalność.
- Inwestorzy naciskają, aby nie było żadnych przerw technicznych w działaniu aplikacji.

Komplikacja



- Gdy liczba użytkowników wzrośnie, nie będziemy w stanie obsłużyć ruchu na serwerze.
- Wdrożenie spowoduje przerwę techniczną.

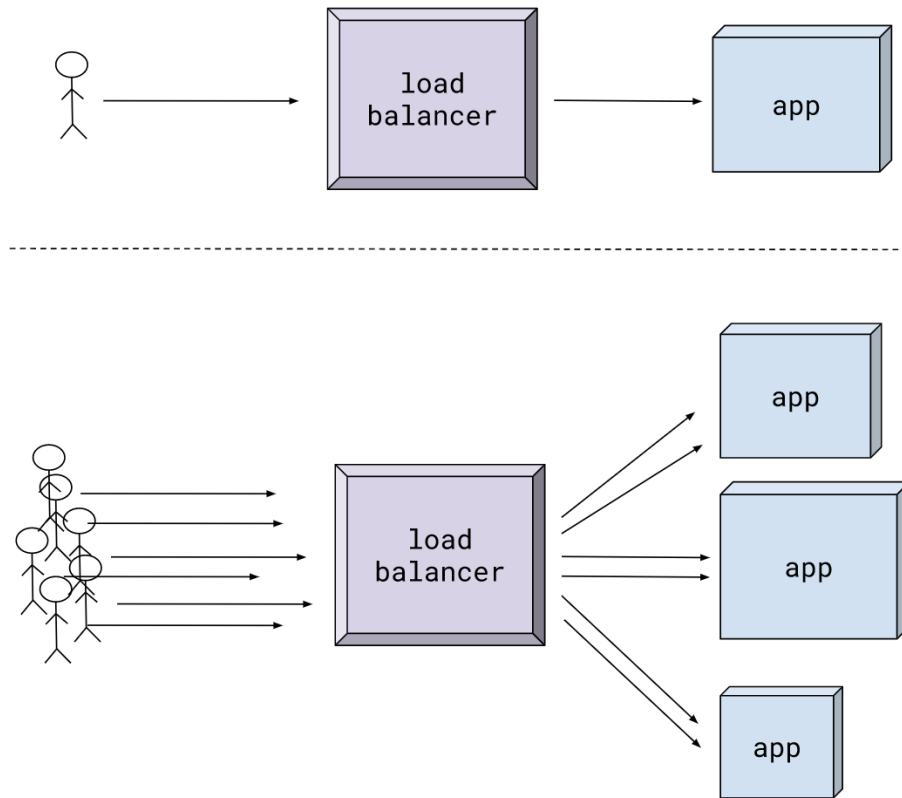
Rozwiązanie - **użycie load balancera**



- Pozwoli nam on wyskalować aplikację horyzontalnie (czyli zwiększając ilość serwerów zamiast wymiany serwera na mocniejszy).
- Umożliwi nam wdrożenie typu blue-green (znane również jako zero-downtime deployment).

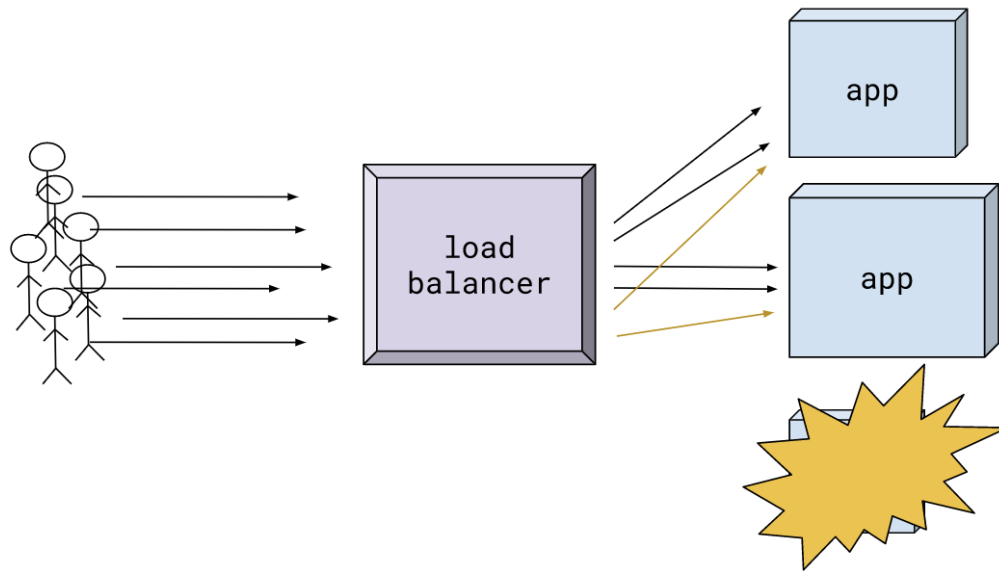
Założenia load balancingu

Skalowalność (ang. Scalability)



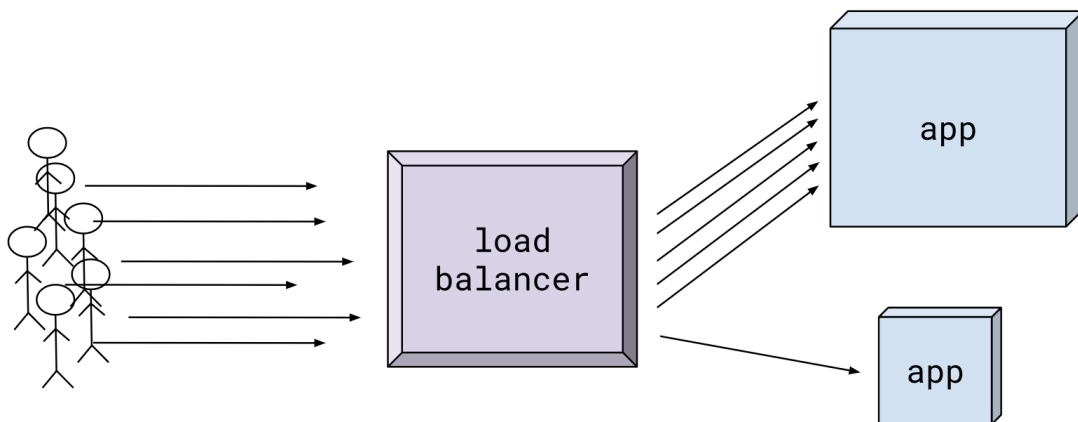
Zdolność dynamicznego przystosowywania się systemu do zwiększonego ruchu bez wpływu na wydajność. Gdy mówimy o load balancingu mamy na myśli zazwyczaj skalowalności horyzontalną, która polega na dodawaniu nowych maszyn/serwerów do klastra obsługującego ruch. Z perspektywy użytkownika proces skalowania powinien być niewidoczny - cały klaster powinien być traktowany jako jeden wirtualny serwer.

Wysoka dostępność (ang. High availability)



O wysokiej dostępności możemy mówić, gdy obsługujemy ruch poprawnie nawet gdy część systemu ulega awarii. Przykładowo, gdy jeden z serwerów ulegnie uszkodzeniu ruch powinien być przekierowany natychmiast do pozostałych.

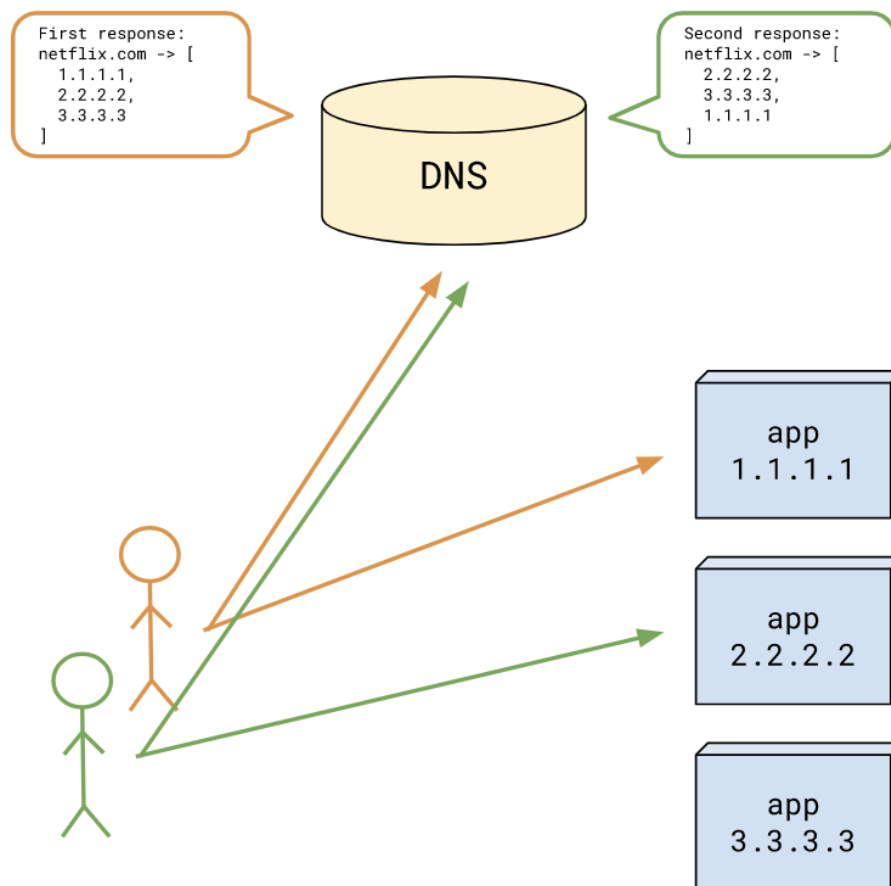
Przewidywalność (ang. Predictability)



Posiadamy kontrolę nad tym jak i kiedy obsłużymy ruch. Oznacza to, że mamy całkowity wpływ na to do którego serwera zostanie przekierowany użytkownik w momencie zapytania.

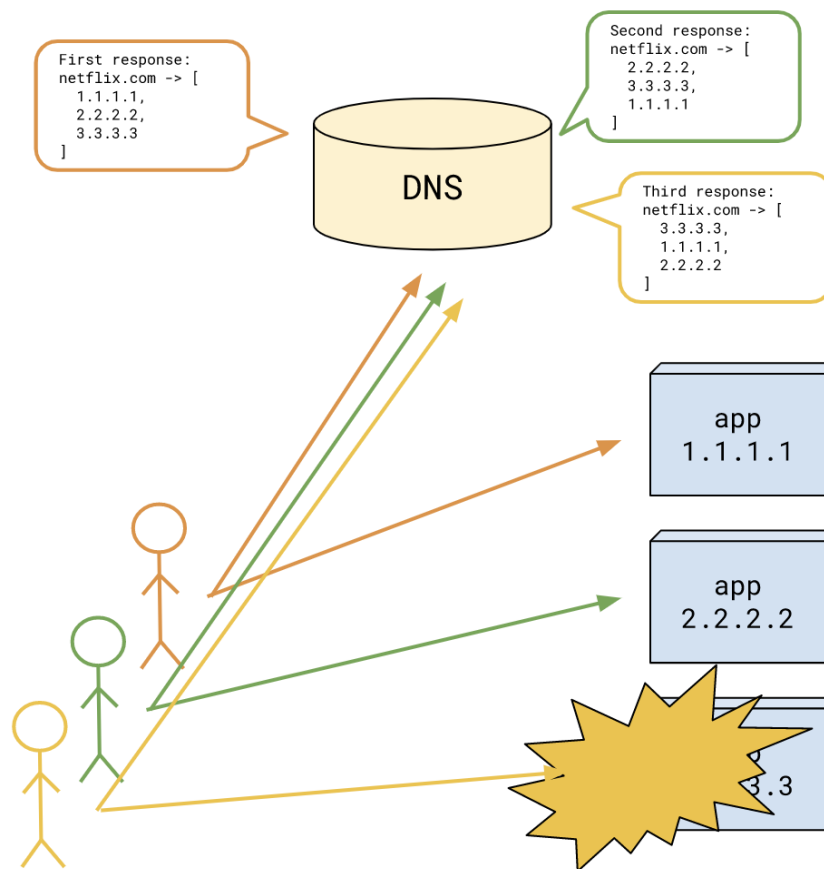
Load balancing z użyciem DNS

Pierwszym rozwiązaniem używanym do podstawowego load balancingu było użycie protokołu DNS. Pozwala on na zwracanie listy adresów gdy pytamy o domenę. Gdy dostajemy taką listę, zazwyczaj wykorzystuje się pierwszy adres do połączenia. DNS zwracając listę adresów w odpowiedzi na kolejne zapytanie wykorzystuje algorytm round-robin. Pierwszy adres z listy zwracany jest na ostatniej pozycji. Dzięki temu kolejna osoba, która odpytuje się DNS, prawdopodobnie wykorzysta inny serwer.



Z tą technologią wiąże się kilka ograniczeń. Pierwsze jest związane z maksymalną długością odpowiedzi jaką może zwrócić serwer DNS. Największa gwarantowana odpowiedź wynosi 512 bajtów¹ (niektórzy klienci obsługują większe, ale nie wszyscy). W Allegro istnieją usługi uruchomione na raz na kilkuset serwerach - to za dużo, aby pomieścić je w dozwolonej odpowiedzi DNS. Ta wada ogranicza nam znacznie skalowalność. Drugą istotną rzeczą jest dostępność. Jeśli jeden z serwerów z listy przestanie działać nie jesteśmy w stanie szybko go z niej wykluczyć.

¹ <https://labs.apnic.net/?p=1380#:~:text=The%20DNS%20operates%20in%20a.DNS%20response%20was%20512%20octets.>



Serwowanie listy adresów nie jest jedynym rozwiązaniem. Posiadając własny serwer DNS (zwany nameserver, NS lub Authoritative DNS) możemy serwować za każdym razem inny adres IP. Możemy dzięki temu przekierować użytkownika do konkretnego serwera, oraz przed udostępnieniem adresu upewnić się, że serwer zlokalizowany pod nim działa poprawnie. Tutaj niestety przeszkodzi nam mechanizm cache'owania odpowiedzi DNS. Zarówno przeglądarka, system operacyjny oraz wszystkie serwery DNS po drodze mogą na jakiś czas zapamiętać serwowany adres. Przez to nie mamy całkowitej kontroli nad tym do jakiego serwera trafi użytkownik i czy będzie on dostępny.

```
A ~ dig +trace allegro.pl
;; Warning: Message parser reports malformed message packet.

: <<> DiG 9.10.6 <<> +trace allegro.pl
;; global options: +cmd
19829 IN NS k.root-servers.net.
19829 IN NS l.root-servers.net.
19829 IN NS m.root-servers.net.
19829 IN NS a.root-servers.net.
19829 IN NS b.root-servers.net.
19829 IN NS c.root-servers.net.
19829 IN NS d.root-servers.net.
19829 IN NS e.root-servers.net.
19829 IN NS f.root-servers.net.
19829 IN NS g.root-servers.net.
19829 IN NS h.root-servers.net.
19829 IN NS i.root-servers.net.
19829 IN NS j.root-servers.net.
;; Received 512 bytes from 197.86.48.46#53(197.86.48.46) in 27 ms

;; Warning: Message parser reports malformed message packet.
pl. 172800 IN NS a-dns.pl.
pl. 172800 IN NS b-dns.pl.
pl. 172800 IN NS c-dns.pl.
pl. 172800 IN NS d-dns.pl.
pl. 172800 IN NS e-dns.pl.
pl. 172800 IN NS f-dns.pl.
pl. 172800 IN NS g-dns.pl.
pl. 172800 IN NS h-dns.pl.
pl. 172800 IN NS i-dns.pl.
pl. 86400 IN DS 38491 8 2 EGEDF662199A448E07BD69BE6CE26902F8D22E5875C9F2305866E05 560549F2
;; Received 512 bytes from 192.58.128.30#53(j.root-servers.net) in 30 ms

;; Warning: Message parser reports malformed message packet.
allegro.pl. 86400 IN NS dns1.allegro.pl.
allegro.pl. 86400 IN NS dns4.allegro.pl.
allegro.pl. 86400 IN NS dns3.allegro.pl.
allegro.pl. 86400 IN NS dns2.allegro.pl.
7U6784FIA3G67151J4TPOF54F304P1UO.pl. 3600 IN NSEC3 1 1 12 7919C37D7A34A317 70675F0C0MP86S19E5J0NCVIP24TOK0I NS SOA TXT RRSIG DNSKEY NSEC3PARAM
7U6784FIA3G67151J4TPOF54F304P1UO.pl. 3600 IN RRSIG NSEC3 8 2 3600 20220624120000 20220424120000 54669 pl. Smjnrho32H8Vh1SkyI8+6iEQNd7YjCLTEVRF4CsWzkkAKB1/R+oBdJk
l Pg8XProulwyZqqZz7FAfdDCRuidcaqcca16Rtuleu2Qz4Ncf+/1kZUHW nnIh7VPPmVt lnPSg4wdmFUBv5M2xxCNzTXB3KnSk+9TodWW/hDnyckE 9GT0108Fk1ls/gpdPmqI2mdhDP07A14EtFPamTlVUTDXj
vjeFnESr2Kj pkHqBhuEfMY80PuojdKIAQamogskDciwXYRTmaNPShSpYufjrsNBveb 3doMNHILRCSzb1QI529RUZbX1bIhPerL46i5R9TR8MrcT7+FCUVYsFmR JIaQ00==
;; Received 512 bytes from 156.154.100.15#53(i-dns.pl) in 45 ms

allegro.pl. 300 IN A 185.31.27.160
;; Received 55 bytes from 2a02:dcc:20::5#53(dns1.allegro.pl) in 31 ms
```

DNS Root Servers

DNS TLD Servers

DNS Allegro Authoritative
Name Servers

Wykorzystując alorytmy DNS doświadczymy również problemu z utrzymaniem przewidywalności. Przez algorytm round robin nie jesteśmy w stanie kontrolować jaka ilość ruchu trafia do konkretnego serwera. Dodatkowo mechanizmy cache'owania mogą sprawić, że ruch ten będzie nierównomierny. To sprawia, że będziemy mieli problemy z przeciążonymi serwerami (w momencie, gdy serwer jest maksymalnie obciążony nadal będą do niego kierowane kolejne zapytania) lub zasoby będą nieefektywnie wykorzystane (zawsze będziemy musieli mieć zapas mocy obliczeniowej na każdym z serwerów).

[Zadanie] DNS

Celem zadania jest dowiedzieć się czy dana usługa korzysta z loadbalancingu za pomocą protokołu DNS. Użyjemy do tego komendy *dig* lub strony <https://www.nslookup.io/>

Wykonajmy proste zapytanie o domenę google.com.

```
λ ~ dig google.com

; <<> DiG 9.10.6 <<> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 18484
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:;, udp: 4096
;; QUESTION SECTION:
;google.com.                IN      A

;; ANSWER SECTION:
google.com.                196     IN      A      216.58.209.14

;; Query time: 16 msec
;; SERVER: 172.20.10.1#53(172.20.10.1)
;; WHEN: Fri Apr 29 16:58:03 CEST 2022
;; MSG SIZE rcvd: 55
```

TTL (Time To Live)

Site IP Address returned by DNS

DNS server address

W odpowiedzi zwróćmy uwagę na:

- Zwrócony adres IP serwera google (216.58.209.14).
- TTL, czyli czas przez jaki jeszcze ten adres będzie ważny (przez tyle sekund możemy go zapamiętać i nie pytać o niego).
- Serwer DNS, który zwrócił nam odpowiedź. W tym przypadku jest to adres sieci lokalnej (172.20.*.*), a więc zwrócony wpis był zapamiętany/zacacheowany przez lokalny DNS resolver

Możemy zapytać o rekordy typu NS - czyli o to jakie serwery odpowiadają za propagowanie domeny google.com.


```

λ ~ dig google.com NS

; <<> DiG 9.10.6 <<> google.com NS
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 45995
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;google.com.                IN      NS

;; ANSWER SECTION:
google.com.                4502    IN      NS      ns3.google.com.
google.com.                4502    IN      NS      ns2.google.com.
google.com.                4502    IN      NS      ns4.google.com.
google.com.                4502    IN      NS      ns1.google.com.

;; Query time: 94 msec
;; SERVER: 172.20.10.1#53(172.20.10.1)
;; WHEN: Fri Apr 29 17:12:39 CEST 2022
;; MSG SIZE rcvd: 111

```

Dostaliśmy adresy czterech serwerów, które są źródłem prawdy na temat tego gdzie znajduje się google.com. Zapytajmy się bezpośrednio jednego z tych serwerów o wpis.

```

λ ~ dig @ns1.google.com google.com

; <<> DiG 9.10.6 <<> @ns1.google.com google.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 63518
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;google.com.                IN      A

;; ANSWER SECTION:
google.com.                300     IN      A      142.250.203.142

;; Query time: 115 msec
;; SERVER: 216.239.32.10#53(216.239.32.10)
;; WHEN: Fri Apr 29 17:15:02 CEST 2022
;; MSG SIZE rcvd: 55

```

Dostaliśmy adres z TTL równym 300 sekund - czyli 5 minut. Przez tyle ten adres może być zapamiętany. Po tym czasie pośrednie serwery DNS muszą się ponownie odpytać jednego z

serwerów NS o adres. Zauważ, że podczas pierwszego zapytania dostaliśmy TTL równy 196. To oznacza, że ten adres będzie zapamiętany na lokalnym DNS jeszcze przez 196 sekund (czyli był prawdopodobnie pobrany z serwera NS 104 sekundy wcześniej).

Zapytanie możemy powtórzyć kilka razy aby sprawdzić, czy za każdym razem dostaniemy ten sam adres. Użyję parametru *+short* dla czytelności (wypisany zostanie sam adres IP).

```
λ ~ dig +short @ns1.google.com google.com
142.250.203.142
λ ~ dig +short @ns1.google.com google.com
142.250.203.142
λ ~ dig +short @ns1.google.com google.com
142.250.203.142
λ ~ dig +short @ns1.google.com google.com
142.250.203.142
λ ~ dig +short @ns1.google.com google.com
142.250.203.142
```

W tym przypadku adres jest jeden, a więc nie możemy potwierdzić, że serwis google.com korzysta z loadbalancingu za pomocą DNS.

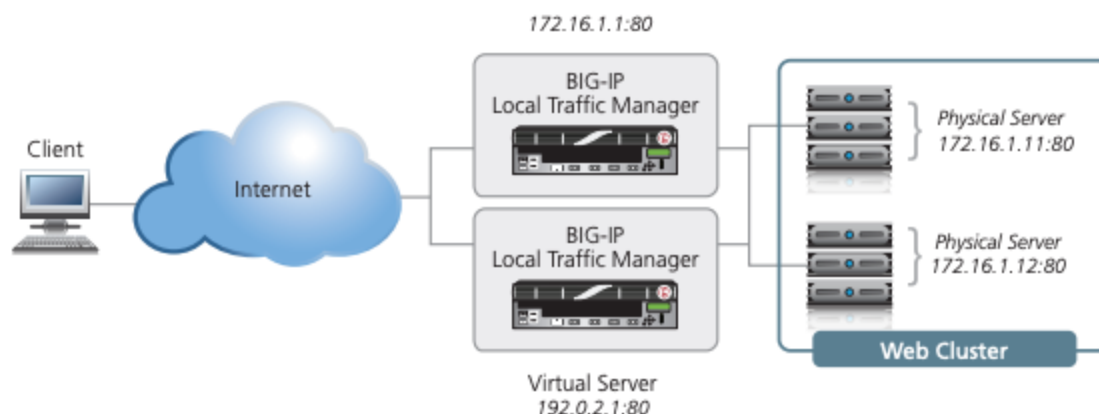
Twoim zadaniem jest wykonanie podobnego testu dla serwisów *allegro.pl* oraz *netflix.com*

1. Jakie są rekordy NS dla danego serwisu?
2. Jaki jest maksymalny TTL dla wpisu z adresem IP?
3. Czy serwis wykorzystuje loadbalancing DNS?

[Dodatkowy materiał] Jak działa DNS możesz dowiedzieć się w tym komiksie:

<https://howdns.works/>

Load balancer jako urządzenie fizyczne



[\(źródło\)](#)

Wykorzystanie fizycznego load balancera pozwala nam całkowicie zwirtualizować nasz serwis dla użytkownika. Nie łączy się on już z konkretną instancją naszej usługi, ale z load balancerem. Zapytanie jest dalej przekazywane do instancji wybranej przez load balancer. Cały proces jest w pełni kontrolowany przez load balancer (wybiera on na jaki konkretny serwer trafi zapytanie). Mając takie rozwiązanie możemy mówić już nie tylko o balansowaniu ruchu (load balancing), ale o jego dystrybucji (load distribution). Load balancer będąc świadomy jakie serwery znajdują się w klastrze może skierować więcej ruchu do tych o większej mocy obliczeniowej i mniej do tych z mniejszą mocą.

Load balancing HTTP

Popularnym modelem działania Load balancera jest load balancing ruchu HTTP. W tym przypadku do load balancera trafia całe zapytanie. Load balancer ma dostęp do metody, ścieżki, nagłówków oraz zawartości zapytania http, jak również zwracanej odpowiedzi. Na podstawie parametrów zapytania mogą podjąć decyzję do jakiego serwera skierować dane zapytanie.

Mikroserwisy i load balancing po stronie klienta

W architekturze mikroserwisowej, gdy robimy zapytanie do innego serwisu (który ma wiele instancji), możemy zaimplementować load balancing po stronie klienta HTTP. Istnieją do tego gotowe rozwiązania (na przykład Spring Cloud LoadBalancer:

<https://spring.io/guides/gs/spring-cloud-loadbalancer/>). Aby zastosować to rozwiązanie potrzebujemy znać również adresy instancji serwisu z którym chcemy się połączyć - ten problem możemy rozwiązać za pomocą Service Discovery.

Algorytmy wykorzystywane do load balancingu

Statyczne

Round robin

Prosty algorytm polegający na przekazywaniu każdego kolejnego zapytania do następnego serwera z listy.

Weighted round robin

Modyfikacja round robin polegająca na przekazywaniu do każdego serwera ilości zapytań proporcjonalnej do jego wagi.

IP hash

Wylicza hash z adresu IP i na jego podstawie wybiera odpowiedni serwer. Gdy serwery obsługujące ruch się nie zmieniają to algorytm ten gwarantuje obsługę zapytań od tego samego użytkownika przez ten sam serwer.

Dynamiczne

Least connection

Wybieramy serwer, który ma najmniej otwartych połączeń.

Weighted least connection

Jak wyżej, ale zakładamy, że niektóre serwery są w stanie obsłużyć więcej połączeń niż inne. Dlatego przypisujemy im wagi.

Weighted response time

Na podstawie ilości otwartych połączeń i średniego czasu odpowiedzi wybieramy serwer, który potencjalnie najszybciej obsłuży zapytanie użytkownika.

Resource based

Wymaga monitorowania zasobów dostępnych w danej chwili na serwerze (zużycie CPU i pamięci). Na tej podstawie wybieramy najmniej obciążony serwer.

Sticky Session

Kiedy chcemy aby sesja danego użytkownika była przekierowana zawsze do tego samego serwera możemy skorzystać z tak zwanego "sticky session". Jest on możliwy do zaimplementowania w Load balancerach HTTP, ponieważ za każdym zapytaniem odczytujemy

nagłówki zapytania - możemy dzięki temu dodawać ciasteczka (cookies) za pomocą których oznaczmy sesję użytkownika.

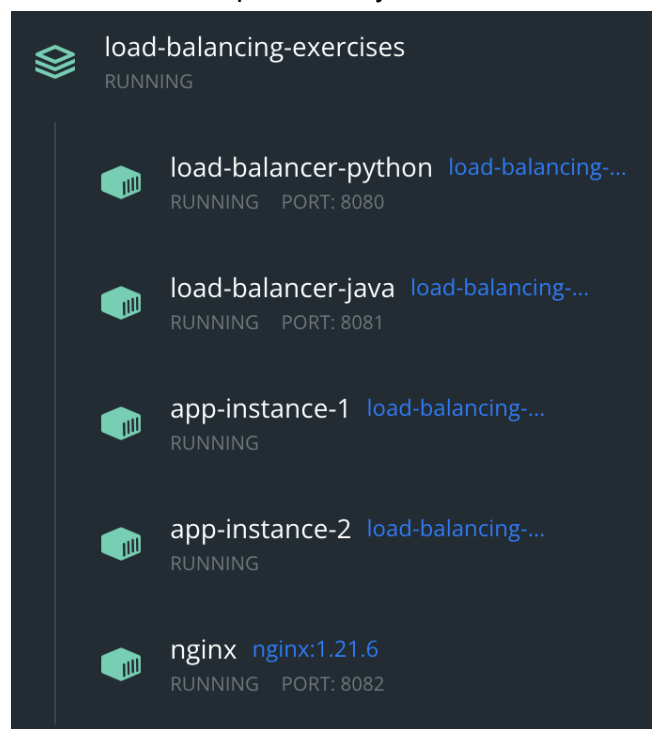
[Zadanie] Load balancing HTTP

Ściągnij na swój komputer repozytorium

<https://github.com/allegro-agh-2022/load-balancing-exercises>

Znajduje się tam kilka kontenerów, które pomogą nam zasymulować infrastrukturę. Uruchomimy je komendą *docker-compose up*.

Po odpaleniu na dashboardzie Dockera powinniśmy widzieć taki zestaw kontenerów:



app-instance-1 oraz *app-instance-2* odpowiadają dwóm serwerom na których działa nasza aplikacja. Implementacja aplikacji znajduje się w katalogu *app/*. Każda z aplikacji działa na jednym wątku, aby w kontrolowanym środowisku lepiej zaobserwować działanie load balancerów. Aplikacja posiada dwa endpointy: *GET /status* oraz *POST /job*. Powinniśmy mieć dostęp do obu endpointów z poziomu load balancera.

load-balancer-python oraz *load-balancer-java* to miejsca na implementacje load balancerów HTTP. W tym ćwiczeniu będziemy pracowali nad ich kodem. Możecie je znaleźć odpowiednio w katalogach *loadbalancer_python/* oraz *loadbalancer_java/*. **Wybierzcie ten język z którym czujecie się lepiej!**

Naszym zadaniem jest napisać prosty loadbalancer HTTP. W repozytorium znajduje się już prosta implementacja - każdy z load balancerów kieruje ruch do jednej instancji.

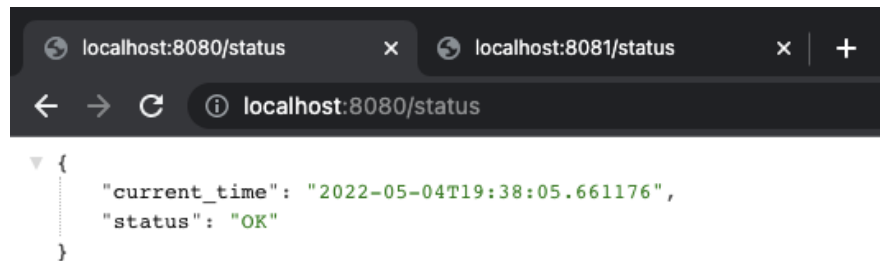
Implementacja load balancera znajduje się odpowiednio w

loadbalancer_python/loadbalancer.py oraz

loadbalancer_java/src/main/java/pl/edu/agh/loadbalancer/LoadbalancerEndpoint.java.

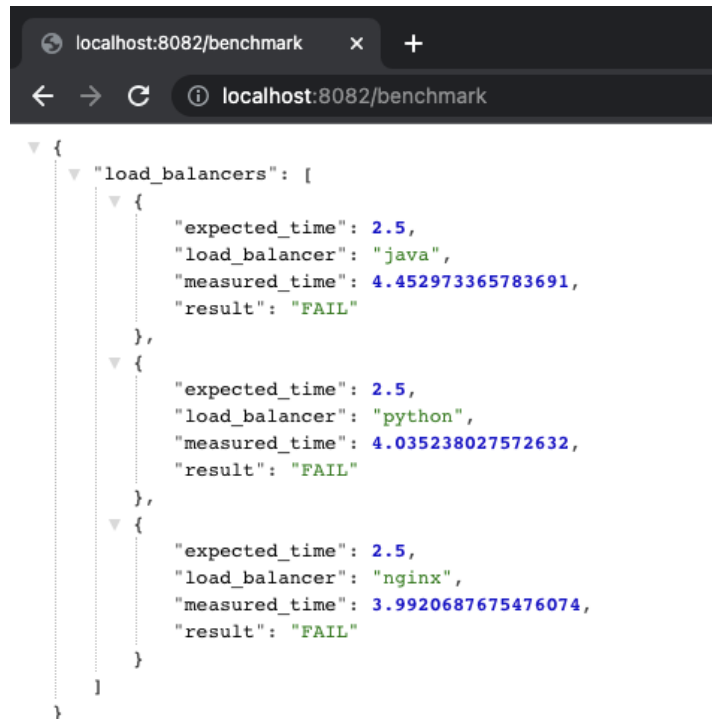
Zorganizuj swój kod wedle uznania - liczy się jakość rozwiązania i efekt końcowy!

Testowanie! W pliku *requests.http* macie zapytania, które możecie uruchamiać z IntelliJ i obserwować w logach jaki jest przepływ zapytania. Endpoint */status* możecie testować również z poziomu przeglądarki (*localhost:8080* to load balancer python, natomiast *localhost:8081* to load balancer java):



Gdy wprowadzisz zmiany w kodzie kontenery musisz przebudować. Najpierw użyj skrótu *Ctrl+C* aby zakończyć działanie kontenerów (w terminalu w którym są odpalone), a następnie użyj komendy: *docker-compose down && docker-compose build && docker-compose up*

Pod adresem <http://localhost:8082/benchmark> znajdziecie również automatyczne testowanie waszych load balancerów - celem jest osiągnięcie statusu "SUCCESS" w jednym z nich.



```
{
  "load_balancers": [
    {
      "expected_time": 2.5,
      "load_balancer": "java",
      "measured_time": 4.452973365783691,
      "result": "FAIL"
    },
    {
      "expected_time": 2.5,
      "load_balancer": "python",
      "measured_time": 4.035238027572632,
      "result": "FAIL"
    },
    {
      "expected_time": 2.5,
      "load_balancer": "nginx",
      "measured_time": 3.9920687675476074,
      "result": "FAIL"
    }
  ]
}
```

Na początek spróbuj rozdzielać ruch pomiędzy instancjami algorytmem Round Robin. Skieruj pierwszy request do *app-instance-1*, drugi do *app-instance-2*, trzeci znowu do *app-instance-1* itd.

Gdy to zrobisz uruchom benchmark (wejdź w przeglądarce na adres *localhost:8082/benchmark*) i zobacz czy rezultat twojego load balancera to "SUCCESS".

Benchmark wysyła do load balancera dwa zapytania. Aplikacje (*app-instance-**) są jednowątkowe, do celów szkoleniowych.

Gdy benchmark dla twojego load balancera przechodzi, spróbuj dopisać kolejne funkcjonalności:

1. Load balancer sprawdza co jakiś czas czy instancja aplikacji jest dostępna (endpoint *GET /status*) i jeśli nie jest to nie kierujemy do niej ruchu. Możesz przetestować to zatrzymując jeden z kontenerów aplikacji, np. *docker stop app-instance-1*.
2. Dodajemy do load balancera endpoint administratora do ustawiania wag instancji. Implementujemy algorytm Weighted Round Robin - każda instancja powinna dostawać ilość ruchu proporcjonalną do swojej wagi.

Na koniec spróbuj skonfigurować loadbalancer używając kontenera Nginx. Instrukcje znajdziesz na tej stronie: <https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/> Skonfiguruj Sticky Session (niestety jest ono dostępne tylko w płatnej wersji Nginx, wykorzystaj algorytm ip hash oraz hash) oraz health check.

Warstwy sieciowe, a loadbalancing

Implementując loadbalancer istotne jest, aby zdecydować na jakiej warstwie sieciowej powinien on działać. Rozróżniamy dwa typy load balancerów w zależności od tego w jakim modelu sieciowym działają.

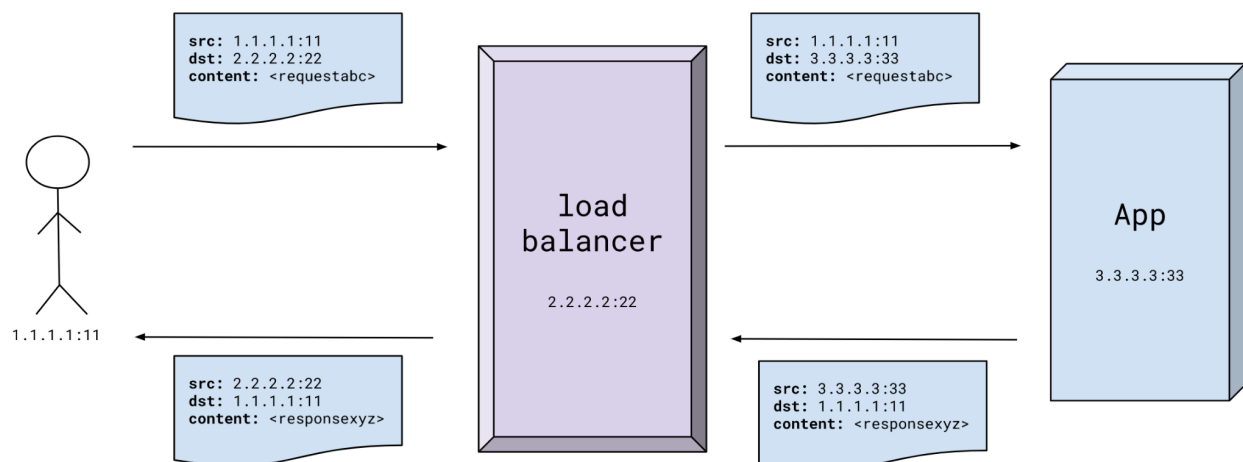
1. Warstwa 4 - transportowa.

Gdy loadbalancer dostaje pakiet warstwy 4 ma do dyspozycji trzy wartości:

- Adres i port źródłowy
- Adres i port docelowy
- Zawartość pakietu (zazwyczaj zaszyfrowana)

Load balancer używa procesu NAT (Network Address Translation) do wykonania dwóch operacji:

- Gdy przychodzi zapytanie to podmienia Adres i port docelowy na dane konkretnej instancji serwera
- Gdy odbiera odpowiedź to podmienia Adres i port źródłowy na swój własny



W takim modelu load balancer nie jest świadomy jaka jest zawartość pakietu, którą przekazuje (może ją odczytać, ale może być ona zaszyfrowana).

2. Warstwa 7 - aplikacji.

Przykładem load balancera działającego na warstwie 7 jest Load balancer HTTP. Load balancer działając na warstwie 7 ma możliwość odczytania zawartości, którą przekazuje. Na podstawie parametrów zapytania może on podjąć decyzję do którego serwera prześle dane zapytanie. W porównaniu do load balancerów działających na warstwie 4, potrzebuje on więcej mocy obliczeniowej (rozszyfrowanie oraz odczytanie zapytania, zaszyfrowanie i odczytanie odpowiedzi, podjęcie bardziej złożonej decyzji). Jest jednak w stanie efektywniej zarządzać ruchem, ponieważ na więcej dostępnych danych na podstawie których rozdziela ruch.

[Zadanie] Load balancing TCP

Ustawić nginx jako loadbalancer TCP:

<https://docs.nginx.com/nginx/admin-guide/load-balancer/tcp-udp-load-balancer/>

Porównać wydajność z loadbalancerami HTTP. Możesz w prostej wersji postawić dwa nginxy - jeden HTTP, jeden TCP i puścić benchmark kilka razy :)

Źródła

<https://www.f5.com/content/dam/f5/corp/global/pdf/white-papers/evolution-adc-wp.pdf>

<https://www.cloudzero.com/blog/horizontal-vs-vertical-scaling>

<https://www.nginx.com/resources/glossary/dns-load-balancing/>

https://en.wikipedia.org/wiki/Round-robin_DNS

<https://labs.apnic.net/?p=1380#:~:text=The%20DNS%20operates%20in%20a,DNS%20response%20was%20512%20octets.>

<https://www.nginx.com/resources/glossary/load-balancing/>

<https://howdns.works/>

<https://www.cloudflare.com/en-gb/learning/performance/types-of-load-balancing-algorithms/>

<https://docs.aws.amazon.com/elasticloadbalancing/latest/application/sticky-sessions.html>

Rysunki