Uniwersytet Technologiczno – Przyrodniczy

w Bydgoszczy

Wydziału Telekomunikacji, Informatyki i Elektrotechniki

Streszczenie rozprawy doktorskiej

mgr inż. Sebastian Łaskawiec

Efektywne rozwiązania dla wysokowydajnej komunikacji w chmurach obliczeniowych

Promotorzy
dr hab. inż. Michał Choraś
dr hab. inż. Tomasz Andrysiak

Bydgoszcz 2020

Streszczenie

Wiele nowoczesnych systemów informatycznych opartych jest o trzy podstawowe komponenty - bazę danych, bezstanowe komponenty realizujące logikę biznesową oraz stronę internetową, która jest interfejsem użytkownika. Komponenty chmur opartych o kontenery wspierają ten model wytwarzania systemów. Istnieje również grupa aplikacji wymykających się wcześniej wspomnianemu podejściu - są to systemy przechowujące dane, w tym systemy typu “Data Grid”. Systemy te wymagają wykonywania dodatkowych czynności przy utrzymaniu systemu, takich, jak monitorowanie obciążenia systemu, konfiguracja wykonywania kopii zapasowych. Autor zaproponował cztery rozwiązania, które rozwiązują część z wcześniej wymienionych problemów poprzez: obniżenie ilości konsumowanej pamięci przez serwer systemu “Data Grid“ (wykorzystując obsługę wielu aplikacji klienckich), zwiększając przepustowość pomiędzy aplikacją kliencką, a serwerem wykorzystując binarne protokoły komuikacyjne oraz technikę ”client side load balancing“ oraz przedstawiając system ekspercki automatycznie znajdujący typowe błędy konfiguracyjne aplikacji. Wszystkie proponowane rozwiązania pomagają zwiększyć przepustowość systemu oraz pomagają zidentyfikować w sposób automatyczny błędy w konfiguracji.

Słowa kluczowe: *Chmury obliczeniowe, Kubernetes, Data Grid*

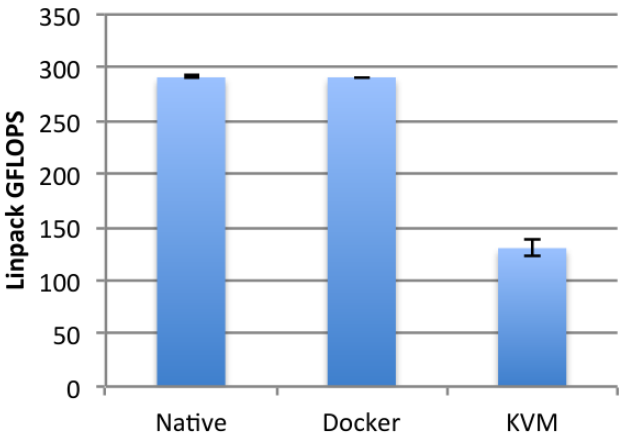
Spis treści

Rozdział 1

Introduction

1.1 Nowoczesne chmury obliczeniowe

Znane firmy, takie, jak Amazon, Apple czy Facebook oferują swoje usługi w każdym miejscu na świecie dzięki dynamicznej infrastrukturze jaką oferuje chmura. W 2006 r. Amazon przedstawił projekt pod nazwą Elastic Compute Cloud Beta, dziś znany jako Amazon EC2. Była to pierwsza tego typu usługa oferowaną szerokiej grupie odbiorców. Podobne podejście zaczęto stosować w projektach typu OpenSource, między innymi w powstałym w 2010 r. projekcie o nazwie OpenStack. Jednakże, wielu programistów zauważyło, że uruchamianie wielu aplikacji na tej samej maszynie wirtualnej często prowadzi do konfliktów między wersjami bibliotek. Ten dość specyficzny problem został rozwiązany przez projekt Linux Containers (LXC), który został spopularyzowany przez firmę Docker w 2013 r. Kontenery umożliwiają pakowanie aplikacji wraz ze wszystkimi zależnymi bibliotekami i systemem operacyjnym. Po spakowaniu, aplikacja przechowywana jest w niezmiennym obrazie i może zostać uruchomiona na maszynie hosta (zazwyczaj jako jeden z procesów w systemie Linuxie). Open Containers Initiative (OCI) przenosi kontenery o krok dalej i standaryzuje interfejsy API używane do budowania, przechowywania i uruchamiania obrazów kontenerów. Utrzymanie takiego ekosystemu różnych aplikacji, które działają na wielką skalę jest dużym wyzwaniem. Opierając się na wcześniejszych doświadczeniach z projektem Borg, firma Google w 2014 r. zapoczątkowała projekt typu OpenSource do koordynowania pracy kontenerów o nazwie Kubernetes. Obecnie Kubernetes jest najpopularniejszą chmurą zorientowaną



Rysunek 1.1: Porównanie wydajności wirtualnych maszyn oraz kontenerów

na kontenery. Jedną z podstawowych zalet takich chmur jest ich wydajność. Rysunek 1.1 przedstawia porównanie wydajności wirtualnych maszyn oraz kontenerów.

Zarządzanie ekosystemem aplikacji w chmurze wiąże się z koniecznością zastosowania orkiestracji. W tej przestrzeni istnieje kilka konkurujących rozwiązań, między innymi Mesos, Docker Swarm i Kubernetes. W chwili pisania niniejszej pracy, Kubernetes jest najbardziej popularnym rozwiązaniem tego typu.

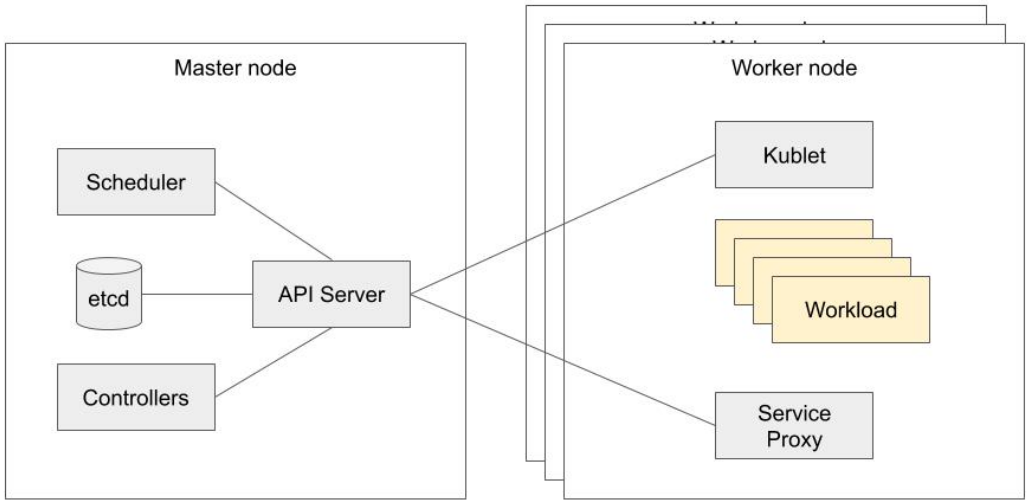
Typowa chmura oparta o projekt Kubernetes składa się z kilku maszyn roboczych oraz znacznie mniejszej liczby maszyn nadzorujących. Przykładowa architektura systemu została przedstawiona na Rysunku 1.2.

Proces nadzorujący składa się poniższych komponentów:

- Scheduler - odpowiedzialny za przypisanie uruchamianej aplikacji do maszyny roboczej
- Etcd - rozproszona baza danych przechowująca konfigurację
- API Server - wykorzystywany do modyfikacji obiektów reprezentujących aplikacje
- Controllers - implementacja logiki wykorzystywana przy uruchamianiu aplikacji w chmurze

Proces maszyny roboczej składa się z:

- Service Proxy - wykorzystywany do wewnętrznej komunikacji
- Kublet - proces nadzorujący



Rysunek 1.2: Przykładowa architektura systemu

- Workload - aplikacje stworzone przez zespoły programistyczne

Kubernetes umożliwia uruchamianie aplikacji z wykorzystaniem chmur różnych dostawców. W wielu przypadkach programiści nie dostrzegą różnic w uruchamianiu aplikacji wykorzystując chmurę Amazon AWS, Microsoft Azure, czy Google Compute Cloud. Ta własność często jest nazywana Cloud Native i została wykorzystana jako podwaliny do stworzenia fundacji "Cloud Native Computing Foundation" (CNCF).

1.2 Aktualne wyzwania w środowiskach chmurowych

W ostatnich latach, uruchamianie systemów przy wykorzystaniu chmur obliczeniowych jest wdrażane w wielu komercyjnych przedsiębiorstwach. Każda z branż ma swoje wymagania i ograniczenia dotyczące modelu wdrożeniowego oraz operacyjnego dla środowisk chmurowych. Wiele z tych wyzwań może być uogólnionych do poniższych grup:

- Wdrażanie aplikacji
- Uwierzytelnienia i autoryzacja (zarówno użytkowników, jak i serwisów pomiędzy sobą)
- Zarządzanie aplikacjami stanowymi

- Informacje audytowe
- Skalowanie aplikacji
- Zarządzanie konfiguracją (zarówno infrastruktury, jak i samej aplikacji)
- Połączenia sieciowe
- Obsługa wielu użytkowników systemu

Istnieje również specjalna grupa wyzwań związana z bezpieczeństwem. Raport firmy StackRox wskazuje, że wdrożenie ponad połowy systemów było opóźnione z powodu incydentów związanych z szeroko pojętym bezpieczeństwem. Wiele z nich było związanych z niepoprawną konfiguracją któregoś z komponentów (samej aplikacji lub infrastruktury). Większość wyzwań związanych z bezpieczeństwem może być podzielonych na poniższe grupy:

- Bezpieczeństwo danych w transporcie
- Zabezpieczenie systemu przed atakami typu “side-channel“
- Ochrona danych przed wyciekiem

Wiele typowych problemów, wraz z ich grupami może być przedstawionych za pomocą poniżej tabeli (Tabla 1.1):

Rozwiązania prezentowane w niniejszej pracy dotyczą trzech ostatnich grup. Do pewnego stopnia, powiązane są one również z aspektami związanymi z bezpieczeństwem - przede wszystkim uwierzytelnianiem i autoryzacją za pomocą certyfikatów.

Grupa wyzwań	Typowy przykład
Wdrażanie aplikacji	Wdrażanie wielu instancji aplikacji
Uwierzytelnienia i autoryzacja	Uwierzytelnianie i autoryzacja użytkowników i usług
Zarządzanie aplikacjami stanowymi	Trwałe zapisywanie danych
Informacje audytowe	Śledzenie zmian w konfiguracji aplikacji
Skalowanie aplikacji	Automatyczne skalowanie aplikacji ze względu na zapotrzebowanie
Zarządzanie konfiguracją aplikacji	**Zarządzanie konfiguracją aplikacji**
Połączenia sieciowe	**Wykorzystywanie protokołów binarnych**
Obsługa wielu użytkowników systemu	**Izolacja danych pomiędzy użytkownikami**

Tabela 1.1: Aktualne wyzwania w środowiskach chmurowych

Rozdział 2

Cele i teza pracy

Głównym celem niniejszej pracy jest zaproponowanie nowych rozwiązań dla polepszenia przepustowości systemu uruchomionego w chmurze obliczeniowej.

Zadania, jakie postawiono, aby osiągnąć powyższy cel są następujące:

1. Zaproponowanie nowych metod izolowania danych pomiędzy użytkownikami systemu i zmniejszenie zapotrzebowania na pamięć serwera systemu typu Data Grid. Pomocniczym zadaniem jest zaprojektowanie nowej metody wykorzystującej właściwości połączeń TCP do zidentyfikowania poszczególnych aplikacji klienckich oraz zaprojektowanie nowej metody dostępu do usługi uruchomionej w chmurze z zewnątrz.

2. Zaproponowanie nowych metod dla udostępniania aplikacji uruchomionej w chmurze dla zewnętrznych aplikacji klienckich wykorzystujących technikę "Client Side Load Balancing".

3. Zaprojektowanie nowego rozwiązania do zmieniania protokołów komunikacyjnych dla aplikacji uruchomionych w chmurze. Zadanie to wymaga wykorzystania nowych metod przełączania komunikacji na protokoły binarne wykorzystujące to samo połączenie TCP.

4. Zaproponowanie nowych rozwiązań do zarządzania konfiguracją aplikacji i odnajdywaniem błędów konfiguracyjnych. Zaproponowane rozwiązanie powinno umożliwić osiągnięcie lepszej wydajności, a także niższego zapotrzebowanie na pamięć systemu.

Teza pracy została sformowana w sposób następujący:

Istnieje możliwość poprawienia szybkości komunikacji, zdefiniowanej jako przepustowość lub opóźnienie, oraz obniżenie zapotrzebowania na pamięć przez serwer systemu typu "Data Grid" poprzez wykorzystanie nowych rozwiązań i algorytmów negocjacji protokołów oraz rozwiązań wykorzystujących technikę "Client Side Load Balancing" dla aplikacji uruchomionych w chmurze.

Rozdział 3

Proponowane rozwiązanie dla identyfikacji aplikacji klienckiej za pomocą pola hostname z rozszerzenia SNI do protokołu TLS

3.1 Wstęp

Wraz z rosnącą popularnością platform typu Infrastructure as a Service (IaaS), twórcy oprogramowania zaczęli eksperymentować z udostępnieniem pojedynczej instancji swojej usługi większej ilości klientów. Takie podejście jest często nazywane ”Multi-tenancy“.

Podstawowym problemem jest separacja danych pomiędzy aplikacje klienckie oraz poprawne ich uwierzytelnienie i autoryzacja. Wiele serwisów wykorzystuje interfejsy REST oparte o protokół HTTP, ale wysokowydajne aplikacje częściej bazują na własnych protokołach binarnych opartych o UDP lub TCP. Rozszerzenie Service Name Indication (SNI) protokołu TLS umożliwia przesłanie do serwera dodatkowego pola o nazwie ”hostname“ w pakiecie transportowanych danych. Pole to jest często wykorzystywane do rozróżniania wirtualnego serwera, który ma obsłużyć dane żądanie.

Separacja danych pomiędzy różnych użytkowników często następuje w warstwie aplikacji. Typowym przykładem tej strategii są bazy danych, które wykorzystują koncepcję schematu, aby przyporządkować logiczne części zbioru danych ich właścicielom.

Dostawca chmury	Koszt
AWS	78 USD per / rok / GB
Google Cloud	78 USD / rok / GB
Microsoft Azure	64 USD /rok / GB

Tabela 3.1: Koszt pamięci w chmurach

Propozycja zawarta w niniejszej pracy wykorzystuje jednak inne podejście - mianowicie wykorzystuje warstwę transportu do identyfikacji poszczególnych aplikacji klienckich.

Protokół TLS (Transport Layer Security) jest odpowiedzialny za nawiązanie szyfrowanego połączenia pomiędzy serwerem a aplikacji kliencką. Aby połączenie mogło zostać zaszyfrowane, niezbędne jest wykonanie procedury "Handshake", którą można podzielić na niżej wymienione kroki:

1. Negocjacja wykorzystywanego mechanizmu szyfrowania
2. Uwierzytelnienie
3. Wymiana kluczy szyfrujących

Rysunek **??** przedstawia schemat procedury nawiązania połączenia za pomocą protokołu TLS wraz z rozszerzeniem SNI.

Rynek oprogramowania typu Data Grid jest bardzo konkurencyjny. Producenci prześcigają się w stworzeniu jak najszybszego rozwiązania wymagającego jak najmniej pamięci. Efektywne zarządzanie zużyciem pamięci jest często czynnikiem decydującym o wdrożeniu konkretnego rozwiązania do większego systemu. Tabla 3.1 przedstawia roczny koszt 1 GB pamięci RAM dla wirtualnej maszyny uruchomionej w chmurze. W przypadku rozwiązania opartego o system typu Data Grid, gdzie większość zbioru danych jest przechowywany w pamięci operacyjnej, te koszty mogą okazać się bardzo wysokie.

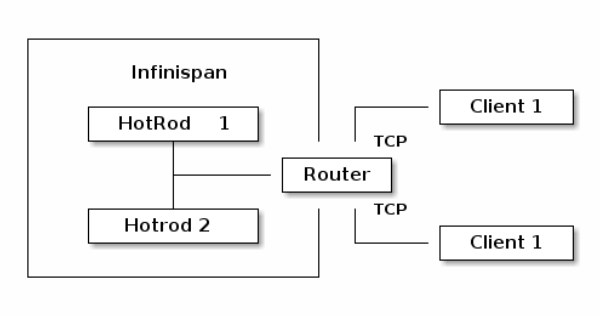
3.2 Proponowane rozwiązanie dla identyfikacji aplikacji klienckiej za pomocą pola hostname z rozszerzenia SNI do protokołu TLS

Celem proponowanego rozwiązania jest opracowanie mechanizmu współużytkowania systemu typu Data Grid w celu obniżenia zapotrzebowania na pamięć przypadającego na pojedynczego użytkownika systemu. Rozwiązanie zostało przetestowane z wykorzystaniem dedykowanego środowiska testowego i założone cele zostały pomyślnie zweryfikowane.

We współdzielonych systemach typu Data Grid, uwierzytelnienie i autoryzacja aplikacji klienckich są szczególnie istotne. Podstawowym zadaniem obu mechanizmów jest upewnienie się, że dane użytkowników są w odpowiedni sposób chronione, a dostęp do nich może być udzielony z wykorzystaniem odpowiednich mechanizmów autoryzacyjnych. Takie mechanizmy mogą być zaimplementowane w warstwie aplikacji lub w warstwie transportu.

W systemach typu Data Grid może istnieć potrzeba konfiguracji parametrów transportowych dla każdej aplikacji klienckiej osobno. Typowym przykładem są dwie aplikacje, gdzie jedna przesyła znikomą ilość dużych obiektów, natomiast druga odwrotnie - przesyła dużą ilość małych obiektów. Konfiguracja buforów zarówno odbiorczych, jak i nadawczych po stronie serwera dla obu przypadków powinna być różna. Podobne kryteria można zastosować do uwierzytelnienia i autoryzacji - różne aplikacje klienckie mogą wymagać różnej konfiguracji. Największą elastyczność można osiągnąć aplikując jak najwięcej ustawień do warstwy transportu, a nie do warstwy aplikacji. Wówczas pole ”hostname“ z rozszerzenia SNI protokołu TLS może być użyte do rozróżnienia aplikacji klienckich i na jego podstawie, serwer wybierze odpowiedni klucz prywatny do zaszyfrowania kanału komunikacyjnego. Odpowiedni klucz zostanie również użyty do przydzielenia dostępu do odpowiedniego segmentu danych zapewniając w ten sposób izolację kontenerów danych różnych użytkowników.

Zastosowanie rozszerzenie SNI protokołu TLS umożliwia wykorzystanie publicznie dostępnych komponentów typu ”Reverse Proxy” do obsługi systemu uruchomionego w chmurze. Pole “hostname” (które nie jest zaszyfrowane) jest wykorzystywane zarówno



Rysunek 3.1: "Multi-tenant Router" i wiele kontenerów danych

przez "Reverse Proxy" do routingu, jak i również przez serwer systemu Data Grid, do rozróżnienia aplikacji klienckich.

Proponowane rozwiązanie "Multi-tenant Router" jest odpowiedzialne za izolowanie kontenerów danych przypisanych do konkretnych aplikacji klienckich. Komponent ten jest współdzielony w całym serwerze i pracuje na jednym z portów TCP (Rysunek 3.1).

Komponent "Multi-tenant Router" wykorzystuje koncepcję ścieżek komunikacyjnych. Każda ścieżka składa się ze źródła (RouteSource) i celu (RouteDestination). Pierwszy z komponentów jest odpowiedzialny za rozpoznanie protokołu komunikacyjnego i rozróżnienie aplikacji klienckich. Drugi, za przełączenie do odpowiedniego kontenera danych (Rysunek 3.1). Algorytm został zaprojektowany ze szczególną dbałością o szybkość działania. Jego złożoność obliczeniowa wynosi $O(n)$.

3.3 Wyniki eksperymentów

Implementacja umożliwia wykorzystywanie protokołu TLS z rozszerzeniem SNI przez aplikację kliencką i system typu Data Grid. Tabela 3.2 oraz Tabela 3.3 zawiera wyniki przeprowadzonych eksperymentów dla zestawienia połączeń oraz wykonania 10.000 operacji w różnej konfiguracji całego systemu. Przeprowadzone pomiary wskazują, że wszystkie otrzymane wyniki są istotne statystycznie.

Nowy komponent nieznacząco podnosi zużycie pamięci (RSS - Resident set size). Szczegółowe wyniki zostały zamieszczone w Tabeli 3.4.

```
Input:  Inbound TLS/SNI connection ic
        SNI Handler sh
        Routing table rt
        Connection pipline p
Output: RouteDestination rd

if(sh.handshakeAccepted(ic))
  shn = sh.getHostname()
  Collection<Route> r = rt.getRoutes()
  // Use lazy evaluation for routes
  Route rd = r
      .filter(ri -> ri.source() is SNISource)
      .filter(ri -> ri.sniHostName() == shn)
      .filter(ri -> ri.desitnation() is HotrodDestination)
      .getFirst()
  if(rd == null)
    throw Exception("No Route")
  pipeline.removeHandler(sh)
  pipline.addHandlers(rd.sestination().handlers())
else
  throw Exception("No TLS/SNI Connection")
```

Rysunek 3.2: Algorytm wykorzystywany w komponencie “Multi-tenant Router”

Opis testu	Ilość iteracji	Wynik [ms/op]	± Błąd
1 Serwer bez SNI	31	3.046	0.096
1 Serwer z TLS/SNI	31	12.725	0.379
2 Serwery oraz SNI router	31	13.471	0.449

Tabela 3.2: Nawiązywanie połączenia

Opis testu	Ilość iteracji	Wynik [ms/op]	± Błąd
1 Serwer bez SNI	31	430.075	18.088
1 Serwer z TLS/SNI	31	847.909	34.479
2 Serwery oraz SNI router	31	1022.655	32.316

Tabela 3.3: Przeprowadzenie 10 000 operacji

Opis testu	Zużyta pamięć
Multi-tenancy wyłączony	482 MB
Multi-tenancy włączony	491 MB

Tabela 3.4: Zużycie pamięci dla proponowanego rozwiązania

3.4 Interpretacja wyników

W przypadku aplikacji zorientowanych na dane (takich, jak bazy danych, czy systemy Data Grid), współdzielenie pojedynczej instalacji przez wielu użytkowników jest jednym z najlepszych sposobów na obniżenie zużycia pamięci oraz kosztów utrzymaniowych systemu. Jednakże wiąże się to również z wieloma zagrożeniami, takimi jak potrzeba izolacji danych pomiędzy użytkownikami oraz zabezpieczenia kanału komunikacyjnego (szyfrowanie). Wykorzystanie rozszerzenia SNI protokołu TLS okazało się bardzo przydatne do rozwiązania obu tych problemów.

Wykorzystanie rozszerzenia SNI w proponowanym rozwiązaniu wiąże się z dodatkowym opóźnieniem (1 milisekunda dla nawiązywania połączenia oraz 175 milisekund dla wykonywania 10 000 operacji). W pierwszym przypadku, opóźnienie jest spodziewane i w związku z tym, że jest jednorazowe, nie powoduje żadnych problemów po stronie klienckiej. Dodanie 17 mikrosekund dla każdej operacji może być okazać się zbyt dużym narzutem w przypadku systemów wymagającej dużej przepustowości.

Proponowane rozwiązanie również wymaga dodatkowej pamięci. W testowanych środowisku, dodatkowy narzut wynosił 1.8 procenta. Jednocześnie, współdzieląc serwer pomiędzy większą ilość aplikacji klienckich, zużycie pamięci przypadające na jednego użytkownika spada. W przypadku 2 użytkowników, zużycie wynosi zaledwie 245.5 MB.

Powyższy eksperyment udowadnia, że teza pracy została częściowo potwierdzona. Istnieje możliwość obniżenia zużycia pamięci przypadającego na jedną aplikację kliencką, ale jest to kosztem szybkości działania systemu.

3.5 Ograniczenia

Systemy zaimplementowane przy użyciu języka programowania Java wymagają dużych ilości pamięci w czasie działania. Współdzielenie serwera pomiędzy aplikacje klienckie ma sens jedynie w przypadku dużej ilości użytkowników. Proponowane rozwiązanie nie będzie efektywne w przypadku pojedynczego użytkownika (lub nawet dwóch). Języki takie, jak C/C++ i Go umożliwiają stworzenie systemu typu Data Grid, który będzie dużo efektywniejszy w przypadku małej ilości aplikacji klienckich.

Kolejnym ograniczeniem jest konieczność stosowania protokołu TLS i szyfrowania połączenia między aplikacją kliencką a serwerem. Niektóre aplikacje, zwłaszcza te, które wykorzystują dane publiczne, nie mają takich wymagań. Wykorzystując proponowane rozwiązanie, takie aplikacje również zostałyby dotknięte dodatkowymi opóźnieniami i zmniejszoną przepustowością.

3.6 Dalsze prace

Proponowane rozwiązanie zostało zaproponowane do wdrożenia w systemie FeedHenry. Podstawowym celem było zastąpienie infrastruktury wykorzystującej projekt Redis rozwiązaniem wykorzystującym projekt Infinispan wraz ze współdzieleniem serwera za pomocą TLS/SNI. Podczas oceny działania prototypu, zauważono, że rozwiązanie musi wspierać dynamiczne dodawanie konfiguracji dla nowych aplikacji klienckich.

Proponowane rozwiązanie zostało również zastosowane z produkcie opartym o projekt Infinispan - Red Hat Data Grid. Obecnie rozwiązanie jest wykorzystywane przez dziesiątki klientów na całym świecie.

Rozdział 4

Proponowane rozwiązanie dostępu do aplikacji pracujących w klastrze w chmurze z wykorzystaniem techniki "client side load balancing"

4.1 Wstęp

Nowoczesne chmury przystosowane są dla standardowych aplikacji wykorzystujących serwisy (odpowiedzialne za logikę biznesową), serwer HTTP (odpowiedzialny za interfejs użytkownika) oraz repozytorium danych (najczęściej relacyjną bazę danych). W wielu przypadkach, konieczne jest również zastosowanie zaawansowanego systemu odpowiedzialnego pamięć podręczną (cache). Takie systemy umożliwiają dostęp do danych w stałym czasie o złożoności obliczeniowej równej $O(1)$. Aplikacje klienckie systemów pamięci podręcznej często wykorzystują technikę "Client Side Loadbalancing" i potrafią zidentyfikować, w której maszynie systemu Data Grid znajduje się określona porcja danych. Dzieje się za pośrednictwem algorytmu "Consistent Hash".

W przypadku chmur opartych o kontenery, ruch sieciowy z zewnątrz chmury jest kierowany za pośrednictwem komponentów typu Reverse Proxy lub Load Balancer. Pierwsze ze wspomnianych rozwiązań posiada wiele zaawansowanych rozwiązań skierowanych do aplikacji wykorzystujących protokół HTTP. Obsługa innych protokołów opartych o TCP, czy UDP jest bardzo ograniczona. W przypadku drugiego rozwiązania,

Opis testu	Wynik [ms/test]
System bez komponentu TLS/SNI	430.075
System z komponentem TLS/SNI	847.909

Tabela 4.1: Wykonanie 10 000 operacji z dodatkowym komponentem odpowiedzialnym za kierowanie ruchu sieciowego [**Laskawiec2017**]

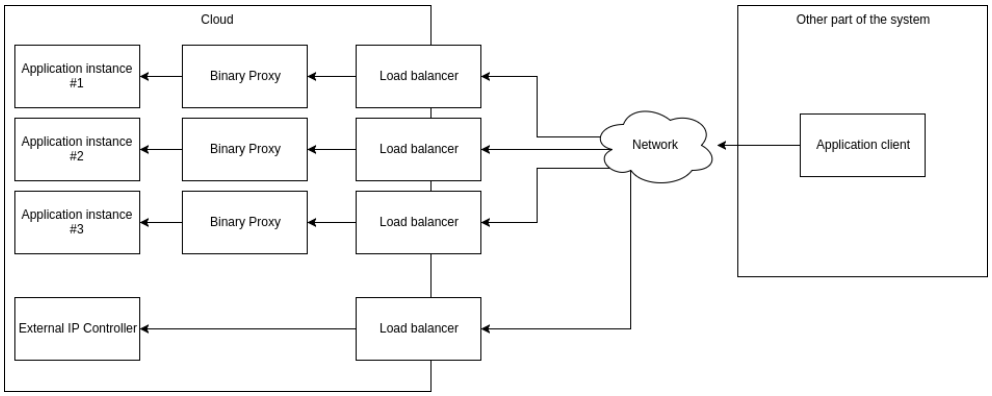
komponentu typu Load Balancer, takich ograniczeń nie ma. W publicznych chmurach wykorzystanie komponentu Reverse Proxy jest najczęściej darmowe, podczas gdy wykorzystanie komponentu Load Balancer jest przeważnie płatne.

Aplikacje klienckie systemów Data Grid, a także wielu aplikacji z branży gier wideo, wykorzystują protokoły binarne oparte o TCP lub UDP. W wielu przypadkach, aplikacje klienckie wymagają skierowania konkretnego żądania do bardzo konkretnego serwera pracującego w klastrze. Rozwiązania prezentowane w niniejszej pracy przedstawiają propozycję rozwiązania tego problemu.

4.2 Proponowane rozwiązanie dostępu do aplikacji pracujących w klastrze w chmurze z wykorzystaniem techniki "client side load balancing"

Celem proponowanego rozwiązania jest zaprojektowanie metody dostępu do konkretnej instancji aplikacji pracującej w klastrze w chmurze ze szczególną dbałością o jak najmniejsze opóźnienie. Rozwiązanie zostało przetestowane z wykorzystaniem dedykowanego środowiska testowego i założone cele zostały pomyślnie zweryfikowane.

Podczas badań, pierwszym z testowanych rozwiązań było zastosowanie rozszerzania SNI protokołu TLS i utworzenie specjalnego komponentu, który dokonywał decyzji, gdzie kierować ruch sieciowy. Niestety, eksperyment ten spowodował spadek wydajności całego systemu (Tabela 4.1).



Rysunek 4.1: The proposed system

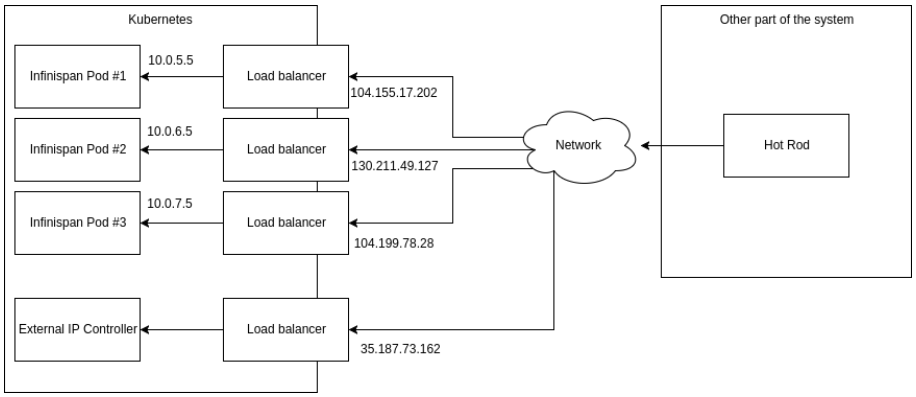
Opis testu	Wykorzystanie Binary Proxy	Wynik [ms/op]
10 000 Operacji	Tak	131.926
10 000 Operacji	Nie	221.711

Tabela 4.2: Testy wydajności komponentu Binary Proxy

Alternatywnym rozważanym rozwiązaniem jest utworzenie własnej implementacji komponentu Load Balancer, który umożliwia inteligentne kierowanie ruchem. Jednakże Load Balancer jest często uważany za element infrastruktury i jest dostarczony przez administratora chmury. Jest to głównym powodem wykluczenia tego rozwiązania z dalszych rozważań.

Ostateczna propozycja rozwiązania została przedstawiona na Rysunku 4.1. Każda z instancji aplikacji pracującej w klastrze posiada przypisany do niej komponent Load Balancer. Dodatkowym testowanym wariantem było umieszczenie komponentu "Binary Proxy", który był odpowiedzialny za separację klastra od świata zewnętrznego. Takie podejście umożliwia bardziej elastyczne skalowanie całego systemu. Niestety, co pokazały testy, również pociąga za sobą spadek wydajności (Tabela 4.2). Dalsze eksperymenty pomijają ten komponent. Komponent "External IP Controller" jest procesem nadzorczym i jego głównym zadaniem jest upewnienie się, czy wszystkie instancje aplikacji mają przypisane odpowiednie komponenty typu Load Balancer i reakcja na zmiany wynikające z dynamicznego skalowania systemu.

Ostateczny zaproponowany system (bez komponentu "Binary Proxy") został przedstawiony na Rysunku 4.2



Rysunek 4.2: Propozycja systemu

```
Input: External address
Output: Internal address

externalAddress = internalAddress
if (needsExternalMapping())
    externalAddress = getExternalAddressFromCache(internalAddress)
    if (externalAddress == null)
        externalAddress = queryExternalIpController(internalAddress)
        putInExternalAddressCache(internalAddress, externalAddress)
return externalAddress
```

Rysunek 4.3: Algorytm zaimplementowany w kliencie systemu Data Grid

Diagram 4.3 przedstawia algorytm zaimplementowany w inteligentnym kliencie systemu Data Grid wykorzystującym technikę "Client Side Load Balancing".

Algorytm komponentu "External IP Controller" został przedstawiony na Diagramie 4.4.

4.3 Wyniki eksperymentów

Testy wydajności zostały przeprowadzone za pomocą aplikacji klienckiej pracującej w dwóch trybach. Tryb "Topology aware" wykorzystuje technikę "Client Side Load Balancing", natomiast tryb "Simple" kieruje żądanie do przypadkowej repliki z listy dostępnych serwerów. Testowane scenariusze sprawdzają opóźnienie wynikające z

```
while (true)
  applicationInstances = queryPlatformForInstances()
  for (applicationInstance : applicationInstances)
      addMarkerLabels(applicationInstance)
      ensureLoadBalancerIsRunning(applicationInstance)
  removeUnnecessaryLoadBalancers()
  sleep(5 min)
```

Rysunek 4.4: Algorytm procesu "External IP Controller"

wykorzystania komponentu Load Balancer (porównanie "L3 + Kubernetes internal" oraz "L3 + Kubernetes internal + LB), sprawdzają różnicę pomiędzy inteligentną a prostą aplikacją kliencką (porównanie "L1 + Kubernetes internal + LB" oraz "L3 + Kubernetes internal + LB") oraz sprawdzają, czy proponowane rozwiązanie poprawia wydajność systemu przy dostępie z zewnątrz chmury (porównanie "L1 + Kubernetes external + LB" oraz "L3 + Kubernetes external + LB"). Wyniki zostały zebrane w Tabeli 4.3.

4.4 Interpretacja wyników

Dostęp do aplikacji pracujących w klastrze uruchomionych w chmurze jest niszowym scenariuszem. Istnieją jednak branże, takie jak inteligentne systemy pamięci podręcznej lub gry wideo, gdzie taka optymalizacja pozwala zbudować systemy o większej przepustowości i oferować usługi na wyższym poziomie.

Proponowane rozwiązanie jest oparte o alokację osobnego komponentu typu Load Balancer dla każdej instancji aplikacji uruchomionej w chmurze. Testy pokazały, że takie podejście umożliwia zastosowanie techniki "Client Side Load Balancing" i zwiększenie przepustowości systemu. Porównując wyniki dla scenariuszy "L3 + Kubernetes internal" oraz "L3 + Kubernetes internal + LB" z Tabeli 4.3, można zauważyć, że wykorzystanie komponentów typu Load Balancer zmniejsza przepustowość o 13.43%. Zastosowanie komponentu Binary Proxy obniża przepustowość aż o 68.05%.

Wyniki pokazują, że zastosowanie proponowanego rozwiązania wraz z inteligentnym klientem pozwala na poprawę o 1%, ale należy mieć na uwadze, że większość czasu

Opis testu	Oznaczenie skrótowe	Wynik [ms/op]	± Błąd
Inteligentny klient uruchomiony wewnątrz chmury	L3 + Kubernetes internal	1288.437	136.207
Inteligentny klient komunikujący się za pomocą komponentu Load Balancer	L3 + Kubernetes internal + LB	1461.510	64.33
Prosty klient komunikujący się za pomocą komponentu Load Balancer	L1 + Kubernetes internal + LB	2163.873	141.332
Inteligentny klient komunikujący się za pomocą komponentu Load Balancer z zewnątrz chmury	L3 + Kubernetes external + LB	2465.586	85.034
Prosty klient komunikujący się za pomocą komponentu Load Balancer z zewnątrz chmury	L1 + Kubernetes external + LB	2684.984	114.993

Tabela 4.3: Wyniki wydajnościowe wykonania 1.000 operacji

potrzebnego na wykonanie testu pochłania opóźnienie między testowanym systemem a maszyną, gdzie została uruchomiona aplikacja kliencka.

Zastosowanie szyfrowania ruchu sieciowego za pomocą protokołu TLS zmniejszyło przepustowość systemu o 30%. Z tego powodu aspekt ten został pominięty w dalszych badaniach.

4.5 Ograniczenia

Alokacja osobnego komponentu typu Load Balancer dla każdej z instancji aplikacji może być bardzo kosztowne w przypadku dużych klastrów. Dodatkowo proponowane rozwiązanie wymaga utrzymania dodatkowego komponentu - "External IP Controller". Komponent ten powinien być wdrożony w sposób odporny na awarie (High Availability).

4.6 Dalsze prace

Pomimo, że wyniki eksperymentu są obiecujące, inne scenariusze wymagają dodatkowych testów, między innymi pomiar czasu dostępu do dysku oraz innych serwisów zależnych od testowanego rozwiązania. Stopień skomplikowania testowanego rozwiązania sprawia, że jest ono skierowane tylko dla systemów szczególnie wymagających wiąże się z dodatkowymi nakładami utrzymaniowymi oraz finansowymi.

Podejście alokacji komponentu typu Load Balancer dla każdej instancji aplikacji zostało również zastosowane w innych rozwiązania, między innymi "Reliable Asynchronous Clustering".

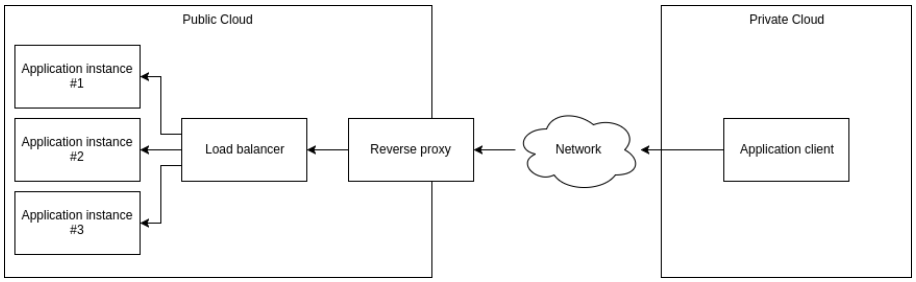
Rozdział 5

Proponowane rozwiązanie dla zmiany protokołów komunikacyjnych

5.1 Wprowadzenie

Nowoczesne metody wytwarzania aplikacji w architekturze opartej o Mikroserwisy bardzo często zakładają uruchamianie usług w chmurze opartej o kontenery. Usługi te przeważnie wykorzystują protokół HTTP oraz interfejsy REST do komunikacji między sobą. Komunikacja pomiędzy usługami zwykle wykorzystuje komponenty typu Load Balancer (oparte o projekt Netfilter, który również jest znany pod nazwą IPTables), natomiast ruch sieciowy z poza chmury jest kierowany do wewnątrz za pomocą komponentu Reverse Proxy (opartego o projekty HAProxy lub Nginx) lub Load Balancer oferowany przez dostawcę chmury (na przykład Elastic Load Balancer w przypadku chmury AWS). Rozwiązania oparte o Reverse Proxy bardzo często są darmowe dla zespołów tworzących aplikacje, ale ogranicza się do protokołu HTTP lub szyfrowanego połączenia za pomocą protokołu TLS (z rozszerzeniem SNI). Wykorzystanie binarnych protokołów sieciowych wiąże się z koniecznością zaalokowania dedykowanego komponentu typu Load Balancer i poniesieniem dodatkowych kosztów (Tabela 5.1).

Większość publicznych chmur wykorzystuje standardowy model przepływu ruchu sieciowego wykorzystujący Reverse Proxy, Load Balancer oraz jedną lub wiele instancji aplikacji (Rysunek 5.1). Typowe aplikacje, zorientowane na wykorzystanie protokołu HTTP i wdrożone w środowiska chmurowe idealnie się wpisują w ten model.



Rysunek 5.1: Model przepływu ruchu sieciowego w chmurze

Dostawca chmury	Koszt
Amazon Web Services Elastic Load Balancer	0.025 USD/h oraz 0.008 USD/GB
Google Cloud Platform Forwarding Rule	0.025 USD/h oraz 0.008 USD/GB

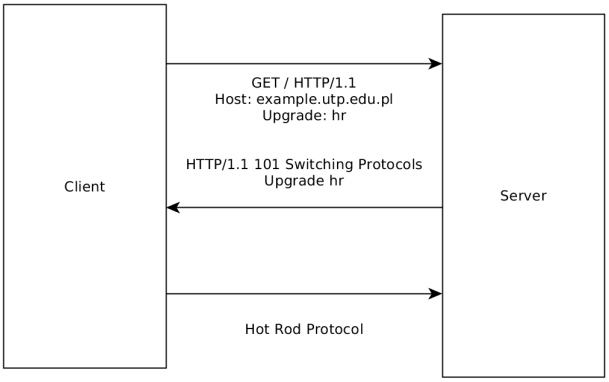
Tabela 5.1: Koszt wykorzystania komponentu typu Load Balancer w chmurze

Istnieją systemy, które zapewniają dostęp do swoich zasobów za pomocą wielu protokołów. Najczęstszą stosowaną techniką jest udostępnianie poszczególnych protokołów na różnych portach TCP lub UDP. Takim przykładem może być serwer Infinispan (Data Grid), który wykorzystuje port 11222 do komunikacji za pomocą protokołu Hotrod i 8080/8443 do komunikacji za pomocą protokołu HTTP (REST). Istnieją również mechanizmy pozwalające na negocjację lub zmianę protokołu komunikacyjnego w locie. Jest to mechanizm HTTP/1.1 Upgrade procedure oraz rozszerzenie ALPN protokołu TLS.

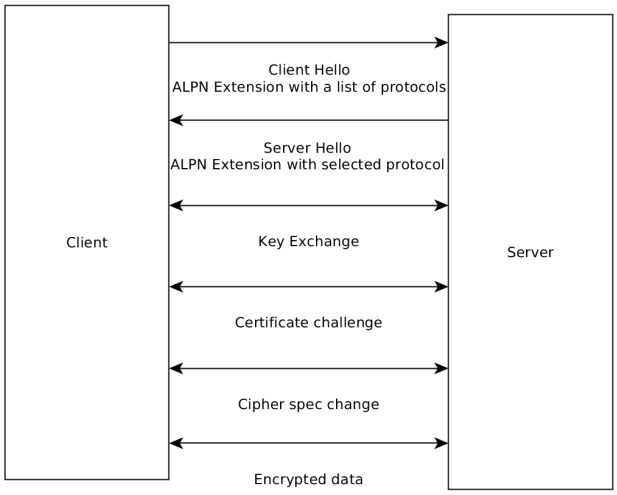
Procedura HTTP/1.1 Upgrade (czasami nazywana "Clear Text Upgrade“) zakłada wysłanie komunikatu HTTP/1.1 101 (Switching Protocols) i odpowiednią reakcję po stronie serwera (co przedstawiono na Rysunku 5.2).

Rozszerzenie "Application-Layer Protocol Negotiation“ (ALPN) protokołu TLS zakłada negocjację protokołu komunikacyjnego podczas procedury "Handshake“ protokołu TLS pomiędzy aplikacją kliencką i serwerem (Rysunek 5.3).

Oba podejścia są wykorzystywane przez przeglądarki internetowe w celu przełączenia na (lub wynegocjowanie) protokołu HTTP/2.



Rysunek 5.2: Procedura HTTP/1.1 Upgrade



Rysunek 5.3: Negocjacja protokołu za pomocą TLS/ALPN

5.2 Proponowane rozwiązanie dla zmiany protokołów komunikacyjnych

Celem proponowanego rozwiązania jest zaprojektowanie mechanizmu przełączania protokołów umożliwiającego wynegocjowanie lub przełączenie na protokół binarny dla aplikacji uruchomionej w chmurze. Proponowane rozwiązanie współpracuje z komponentem Reverse Proxy, którego użycie jest darmowe w większości chmur opartych o kontenery. Rozwiązanie zostało przetestowane z wykorzystaniem dedykowanego środowiska testowego i założone cele zostały pomyślnie zweryfikowane.

Mechanizm wykorzystywany przez rozszerzenie ALPN protokołu TLS, jak i procedurę HTTP/1.1 Upgrade, jest oparty o listę protokołów przesyłaną przez aplikację kliencką do serwera. Serwer może zaakceptować lub odrzucić proponowany protokół komunikacyjny. Podejście to zostało wykorzystane w proponowanym rozwiązaniu i umożliwia oferowanie wiele usług (opartych o różne protokoły) na pojedynczym porcie TCP lub UDP. W obu przypadkach połączenie na warstwie transportu (TCP lub UDP) jest wykorzystywane ponownie i zmianie ulega tylko protokół wyższej warstwy, na przykład z HTTP/1.1 na HTTP/2.

Fakt, że połączenie transportowe nie ulega przełączaniu, umożliwia wykorzystanie komponentu Reverse Proxy do nawiązania połączenia wykorzystującego protokoły binarne. Proponowane rozwiązanie wspiera zarówno negocjację protokołu za pomocą rozszerzenia ALPN, jak i nawiązanie połączenia HTTP i natychmiastową zmiana na protokół binarny.

Proponowane rozwiązanie zostało zaimplementowane w projekcie Infinispan i wykorzystuje 3 protokoły: HTTP/1.1, HTTP/2 oraz Hotrod. Implementacja bazuje na module Multi-tenancy Router zaimplementowanym we wcześniej omówionej propozycji. Algorytm mechanizmu dokonującego negocjacji (oraz przełączania) protokołów został zaprezentowany na Diagramie 5.4.

```
Input:  Inbound connection ic
        Routing table rt
        UpgradeHandler uh
        CommunicationPipeline cp
Output: SinglePortUpgradeHandler rd

Protocol p = uh.negotiate(ic)
Handler h = rt.getHandler(p)
if (h == null)
  h = rt.getHandler("HTTP/1.1")
cp.addHandler(h)
```

Rysunek 5.4: Algorytm przełączający

5.3 Wyniki eksperymentów

Eksperyment przeprowadzono za pomocą inteligentnego klienta protokołu Hotrod wspierającego również protokoły HTTP/1.1 oraz HTTP/2 (zarówno w wariancie wykorzystującym TLS/ALPN, jak i procedurę HTTP/1.1 Upgrade). Do eksperymentów wykorzystano pojedynczą instancję serwera Infinispan oraz oprogramowanie do testów wydajności - JMH. Tabela 5.2 przedstawia zestawienie czasów inicjalizacji połączenia.

Tabela 5.3 oraz Tabela 5.4 przedstawiają testy wydajności dla obiektów zawierających 360 bajtów oraz 36 bajtów danych.

5.4 Interpretacja wyników

Negocjacja i zmiana protokołów komunikacyjnych nie jest łatwym tematem. Wiele języków programowania obsługuje HTTP/1.1 w swoich bibliotekach standardowych, jednak wsparcie dla HTTP/2 lub protokołów binarnych nie jest już tak oczywiste. Możliwość negocjacji protokołu sprawia, że oprogramowanie przechowujące dane (takie, jak systemy Data Grid) są bardziej wszechstronne.

Dodatkowo podejście to umożliwia ponowne wykorzystanie wcześniej nawiązanego połączenia TCP i zmianę jedynie protokołu wyższej warstwy (na przykład z HTTP/1.1

Mechanizm przełączający	Typ połączenia	Protokół	Ilość iteracji	rezultat [ms/op]	± Błąd
Brak	Bezpośrednio	HTTP/1.1	31	2.068	0.686
Brak	OCP Router	HTTP/1.1	31	1.087	0.154
TLS/ALPN	Bezpośrednio	HTTP/2	31	5.063	0.531
TLS/ALPN	OCP Router	HTTP/2	31	6.576	1.535
HTTP/1.1 Upgrade	Bezpośrednio	HTTP/2	31	3.310	0.864
HTTP/1.1 Upgrade	OCP Router	HTTP/2	31	4.464	1.617
TLS/ALPN	Bezpośrednio	Hotrod	31	9.742	1.102
TLS/ALPN	OCP Router	Hotrod	31	10.401	1.302
HTTP/1.1 Upgrade	Bezpośrednio	Hotrod	31	5.389	1.067
HTTP/1.1 Upgrade	OCP Router	Hotrod	31	8.594	10.122

Tabela 5.2: Testy wydajności nawiązywania połączenia

Mechanizm przełączający	Typ połączenia	Protokół	Ilość iteracji	rezultat [ms/op]	± Błąd
None	Bezpośrednio	HTTP/1.1	31	0.472	0.330
None	OCP Router	HTTP/1.1	31	1.315	0.781
TLS/ALPN	Bezpośrednio	HTTP/2	31	1.577	0.282
TLS/ALPN	OCP Router	HTTP/2	31	2.149	0.480
HTTP/1.1 Upgrade	Bezpośrednio	HTTP/2	31	1.048	0.078
HTTP/1.1 Upgrade	OCP Router	HTTP/2	31	1.156	0.078
TLS/ALPN	Bezpośrednio	Hotrod	31	0.269	0.039
TLS/ALPN	OCP Router	Hotrod	31	0.331	0.048
HTTP/1.1 Upgrade	Bezpośrednio	Hotrod	31	0.193	0.037
HTTP/1.1 Upgrade	OCP Router	Hotrod	31	0.255	0.051

Tabela 5.3: Testy wydajności dla 360 bajtów danych

Mechanizm przełączający	Typ połączenia	Protokół	Ilość iteracji	rezultat [ms/op]	± Błąd
Brak	Bezpośrednio	HTTP/1.1	31	0.426	0.337
Brak	OCP Router	HTTP/1.1	31	2.310	0.761
TLS/ALPN	Bezpośrednio	HTTP/2	31	0.754	0.196
TLS/ALPN	OCP Router	HTTP/2	31	0.649	0.100
HTTP/1.1 Upgrade	Bezpośrednio	HTTP/2	31	0.267	0.062
HTTP/1.1 Upgrade	OCP Router	HTTP/2	31	0.305	0.055
TLS/ALPN	Bezpośrednio	Hotrod	31	0.267	0.062
TLS/ALPN	OCP Router	Hotrod	31	0.302	0.052
HTTP/1.1 Upgrade	Bezpośrednio	Hotrod	31	0.231	0.065
HTTP/1.1 Upgrade	OCP Router	Hotrod	31	0.243	0.040

Tabela 5.4: Testy wydajności dla 36 bajtów danych

na HTTP/2). Ta właściwość umożliwia wykorzystanie komponentów typu Reverse Proxy do komunikacji wykorzystującej protokoły binarne, a te oferują znacznie większą przepustowość i możliwości, niż HTTP/1.1, czy nawet HTTP/2.

Wyniki eksperymentów jasno wskazują, że czas potrzebny do nawiązania połączenia w przypadku protokołów innych niż HTTP/1.1 jest znacząco wydłużony. Najdłuższy czas został zaobserwowany dla protokołu binarnego przy wykorzystaniu mechanizmu TLS/ALPN. Jest to wynik spodziewany. Protokół Hotrod podczas nawiązywania połączenia przesyła informacje o topologii klastra serwerów i informacje, w którym z serwerów dane są przechowywane (Consistent Hash).

Testy wydajnościowe, zarówno dla 36, jak i 360 bajtów wskazują szybszy transfer danych za pomocą łącza nieszyfrowanego. Ponownie, jest to rezultat spodziewany. Szyfrowanie danych sprawia, że ilość danych do transmisji jest większa dodatkowo konsumując czas procesora na czynności kryptograficzne. Rezultaty również jasno wskazały, że zastosowanie protokołów binarnych jest znacząco szybsze od HTTP/1.1, czy HTTP/2. Należy również zauważyć, że procedura testowa była synchroniczna. Zarówno aplikacja kliencka, jak i serwer Infinispan obsługują tryb asynchroniczny. Wówczas spodziewany wynik byłby jeszcze lepszy dla protokołów binarnych.

Wyniki potwierdzają tezę stawianą w niniejszej pracy - zastosowanie proponowanego mechanizmu przełączania wspierającego zarówno warianty nieszyfrowane (HTTP/1.1 Upgrade), jak i szyfrowane (TLS/ALPN) pozwalają na osiągnięcie najlepszej wydajności.

Dodatkowo takie rozwiązanie pozwala wykorzystać komponent Reverse Proxy, co również obniża koszty utrzymaniowe systemu.

5.5 Ograniczenia

Jednym z podstawowych ograniczeń mechanizmu przełączającego opartego o TLS/ALPN jest fakt, że protokół komunikacyjny jest negocjowany tylko podczas procedury TLS Handshake. W tym przypadku rozwiązanie oparte o procedurę HTTP/1.1 Upgrade jest bardziej elastyczne i umożliwia rozpoczęcie procedury przełączającej w każdym momencie.

Dodatkowym ograniczeniem jest wykorzystanie protokołu TCP, jako transportu dla protokołów wyższego rzędu. Proponowane rozwiązanie jest oparte o fakt, że raz nawiązane połączenie stanowe TCP jest ponownie wykorzystywane. Bez tego założenia nie można wykorzystać proponowanego rozwiązania w scenariuszu wykorzystującego Reverse Proxy.

5.6 Dalsze prace

Proponowane rozwiązanie zostało zaimplementowane w projekcie Infinispan pod nazwą "Single Port". W czasie pisanie niniejszej pracy dostępny mechanizm przełączający obsługuje protokoły HTTP/1.1, HTTP/2 oraz Hotrod. W przyszłości zostanie on rozszerzony o WebSockets i Memcached.

Kolejnym krokiem jest stworzenie wielo-protokołowej aplikacji klienckiej, która będzie potrafiła dobrać odpowiedni protokół do okoliczności.

Rozdział 6

Proponowane rozwiązanie do automatycznego wykrywania błędów konfiguracyjnych

6.1 Wstęp

Aktualne badania i raporty firmy StackRox wskazują, że niewłaściwa konfiguracja na poziomie zarówno infrastruktury, jak i aplikacji może być przyczyną problemów z wydajnością i bezpieczeństwem systemu informatycznego. Wnioski te potwierdza ilość pytań pojawiająca się na forum społeczności projektów Keycloak i Infinispan (których autor niniejszej pracy jest współautorem).

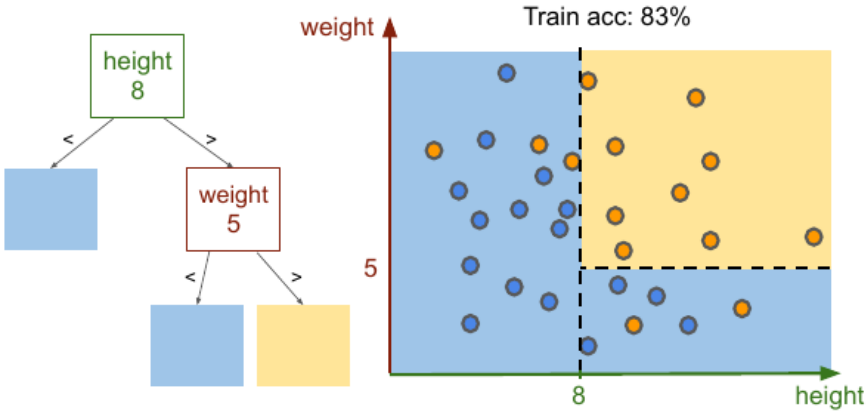
Oprogramowanie umożliwiające automatyczne wykrywanie błędów konfiguracyjnych ułatwiłoby utrzymanie wielu projektów OpenSource oraz pozwoliłoby wielu inżynierom skupić się mocniej na rozwoju ich projektów.

Podstawowym wyzwaniem systemów eksperckich umożliwiających wykrywanie usterek jest klasyfikacja danych wejściowych opisujących system. Rezultatem takich klasyfikacji jest rozpoznanie błędu konfiguracyjnego. Metody Uczenia Maszynowego wykorzystują w tym celu tzw. Klasyfikatory.

Do oceny działania klasyfikatorów wykorzystuje się różne miary, ale jedną z najpopularniejszych jest Celność (Accuracy). Tabela 6.1 przedstawia terminologię pomocniczą (po angielsku) przydatną w obliczaniu Celności (Równanie 6.1).

		Predicted	
		Positive	Negative
Actual	Positive	True Positives (TP)	False Negatives (FN)
	Negative	False Positives (FP)	True Negatives (TN)

Tabela 6.1: Model accuracy measures for classification



Rysunek 6.1: Drzewo decyzyjne

$$Celno = \frac{TP + TN}{TP + TN + FN + FP} \tag{6.1}$$

Jedne z najefektywniejszych klasyfikatorów (wykorzystywanych w modelach, które wygrały wiele konkursów) są oparte o Drzewa Decyzyjne (Rysunek 6.1).

Z praktycznego punktu widzenia, Drzewa Decyzyjne charakteryzują się następującymi właściwościami:

1. Dane wejściowe nie wymagają normalizacji
2. Decyzje podejmowane w ramach drzewa mogą być łatwo wytłumaczone za pomocą grafu
3. Drzewo Decyzyjne nie może przewidzieć danych, których zakres wykracza poza dane treningowe

Rozwiązanie prezentowane w niniejszej pracy wykorzystuje również ustandaryzowany sposób tworzenia własnych zasobów w chmurze opartej o kontenery - Operator Framework. Jest to ekosystem narzędzi, który składa się głównie z obiektu typu Controller,

które reaguje na zmiany w określonych zasobach (tzw. "Custom Resource"). Operator tworzy swego rodzaju interfejs pomiędzy konfiguracją aplikacji, a tym, w jaki sposób jest ona wdrożona do działającego systemu.

Podejście wykorzystujące Operator Framework umożliwia utworzenie oprogramowania automatyzującego szereg administracyjnych czynności wymaganych do utrzymania sprawnie działającego oprogramowania, m.in.:

1. Automatyczna instalacja oprogramowania w chmurze
2. Zarządzanie konfiguracją
3. Automatyczne aktualizowanie wersji
4. Zarządzanie kopiami zapasowymi
5. Monitorowanie aplikacji
6. Automatyczne skalowanie

System do automatycznego wykrywania niepoprawnej konfiguracji jest odmianą Systemu Eksperckiego. Większość aktualnych systemów tworzona jest przy pomocy Silników Reguł lub wykorzystując Uczenie Maszynowe.

6.2 Proponowane rozwiązanie do automatycznego wykrywania błędów konfiguracyjnych

Celem proponowanego rozwiązania jest zaprojektowanie systemu do automatycznego wykrywania niepoprawnej konfiguracji aplikacji uruchomionej w chmurze opartej o kontenery. Rozwiązanie zostało przetestowane z wykorzystaniem dedykowanego środowiska testowego i założone cele zostały pomyślnie zweryfikowane.

Istnieje wiele rozwiązań, takich jak Magalix, czy Red Hat Artificial Intelligence Center of Excellence, które umożliwiają automatyczne wykrywanie błędów w komponentach infrastrukturalnych chmury. Niestety, nie ma obecnie na rynku uniwersalnego narzędzia

skierowanego do aplikacji. Utworzenie takiego rozwiązania wiąże się z wieloma problemami, takimi jak:

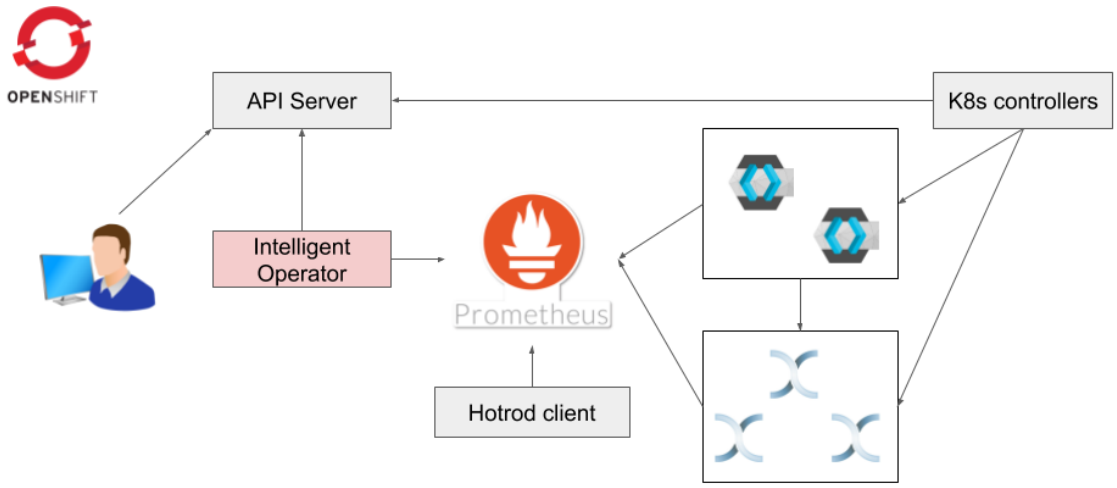
1. Obsługa różnych formatów plików konfiguracyjnych
2. Brak środowiska do trenowania modeli Uczenia Maszynowego i modeli statystycznych
3. Brak standardowej infrastruktury agregującej metryki aplikacji
4. Różnice w środowisku wdrożeniowych pomiędzy aplikacjami

Inicjatywa Operator Framework oraz chmura Kubernetes wiele tych aspektów standaryzują. Różne formaty konfiguracyjne zostały ujednolicone w koncept "Custom Resource". Istnieje wspólna dla całej chmury infrastruktura agregująca metryki aplikacji (projekt Prometheus), a środowisko wdrożeniowe jest zunifikowane.

Celem proponowanego rozwiązania jest utworzenie systemu, który w sposób ujednolicony wykrywa błędy konfiguracyjne w oparciu o inspekcję plików konfiguracyjny i metryk aplikacji. Architektura proponowanego rozwiązania została przedstawiona na Rysunku 6.2. Rozwiązanie zostało nazwane "Inteligentny Operator". Proponowane rozwiązanie odczytuje metryki aplikacji z bazy danych Prometheus oraz pliki konfiguracyjne z serwera API projektu Kubernetes. Projekt zakłada, że każda z uruchomionych aplikacji raportuje metryki związane z jej działaniem do bazy danych Prometheus i wykorzystuje jeden z obiektów przeznaczonych do przechowywania konfiguracji (Custom Resource, Secret lub ConfigMap) w chmurze Kubernetes.

Projekt wykorzystuje Bazę Wiedzy (Knowledge Base) zapisaną przez inżynierów opiekujących się projektem w formacie arkusza kalkulacyjnego. Tabela 6.2 przedstawia jeden z przykładów takiej Bazy Wiedzy skierowany do wykrywania odpowiedniego protokołu wymaganego do działania aplikacji w klastrze (KUBE_PING). Jeśli konfiguracja aplikacji nie posiada odpowiedniego wpisu, zostanie zwrócona sugerowana akcja.

Proponowane rozwiązanie umożliwia import zarówno metryk, jak i konfiguracji do tej samej struktury danych. Podejście to umożliwia wykrywanie błędów związanych z niepoprawnymi zależnościami między konfiguracją a metrykami (Tabela 6.3).



Rysunek 6.2: Architektura proponowanego rozwiązania

up	jgroups_kube_ping_count	ACTION
1	0	There is no KUBE_PING configuration and yet, we are running in Kubernetes

Tabela 6.2: Baza Wiedzy wykrywająca brakujący protokół KUBE_PING w konfiguracji

cluster_lock_timeout	ACTION
metrics.filter(like='infinispan-app', axis=0).loc[:,'connector_replication_timeout'][0]	Cluster lock is too small

Tabela 6.3: Baza Wiedzy do wykrywania niepoprawnej konfiguracji parametru "Cluster Lock"

	up	jgroups_kube_ping_count	jgroups_dns_ping_count	cluster_lock_timeout	jvm_threads_current
0	0.000	nan	nan	NaN	nan
1	nan	0.000	nan	NaN	nan
2	nan	nan	0.000	NaN	nan
3	nan	nan	nan	metrics.filter(like='infinispan-app', axis=0)....	nan
4	nan	nan	nan	NaN	200.000

Tabela 6.4: Wejściowa Baza Wiedzy Inteligentnego Operatora

	up	jgroups_kube_ping_count	jgroups_dns_ping_count	cluster_lock_timeout	jvm_threads_current	ACTION
0	0.000	1.000	0.000	60000	170.889	The service is down. Please make sure it's running
1	1.000	0.000	0.000	60000	170.889	There is no KUBE_PING configuration and yet, we are running in Kubernetes
2	1.000	1.000	0.000	60000	170.889	There is no DNS_PING configuration and yet, we are running in Kubernetes
3	1.000	1.000	0.000	5000	170.889	Cluster lock is too small
4	1.000	1.000	0.000	60000	200.000	Lower the number of worker threads

Tabela 6.5: Sparsowana Baza Wiedzy Inteligentnego Operatora

6.3 Wyniki Eksperymentów

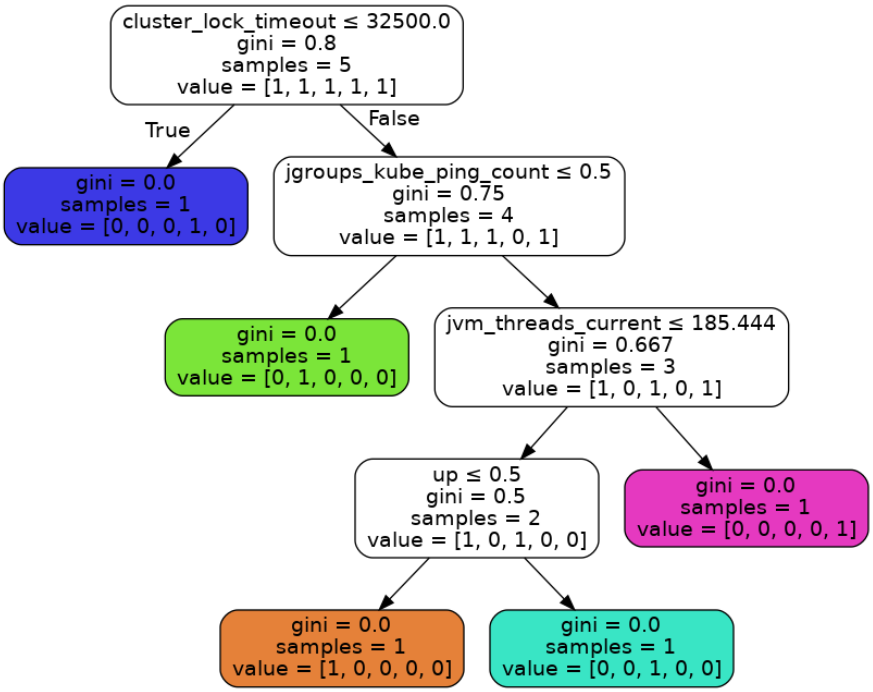
Eksperymenty zostały przeprowadzone z wykorzystaniem uproszczonej wersji Bazy Wiedzy przechowywanej w arkuszu kalkulacyjnym. Raz zapisana Baza Wiedzy, została zaimportowana przez system wykorzystujący Operator Framework. Dane wejściowe zostały zaprezentowane w Tabeli 6.4. Tabela 6.5 przedstawia sparsowane dane odczytane przez system.

W następnym kroku system na podstawie danych wejściowych utworzył model klasyfikatora. W przypadku proponowanego systemu jest to Drzewo Decyzyjne, które zostało przedstawione na Rysunku 6.3.

Celność modelu jest równa 1.

6.4 Interpretacja wyników

Proponowane rozwiązanie zostało przetestowane zarówno przez osoby utrzymujące projekty OpenSource, jak i przez obsługę techniczną w firmie Red Hat. Prototyp okazał



Rysunek 6.3: Decision boundaries of a Decision Tree Classifier

się niezwykle pomocny w przypadku nowych instalacji, gdzie wiele komponentów jest modyfikowanych w tym samym czasie.

Testowany model poprawnie zwraca rekomendowane akcje wyuczone w czasie treningu. Jednakże pisanie i utrzymywanie Bazy Wiedzy wymaga sporo wysiłku.

Decyzje podejmowane przez model zaimplementowany w proponowanym rozwiązaniu mogą zostać łatwo wytłumaczone na podstawie grafu. Jest to szczególnie istotne w przypadku obsługi rozwiązania przez wykwalifikowaną kadrę.

Pomimo, że proponowane rozwiązanie nie dowodzi poprawności tezy niniejszej pracy w sposób bezpośredni, utrzymywanie poprawnej konfiguracji aplikacji uruchomionej w chmurze jest szczególnie istotne i wprost wymagane w celu utrzymania wysokiej wydajności systemu.

6.4.1 Ograniczenia

Proponowane rozwiązanie jest zaledwie prototypem większego systemu, który być może będzie rozwijany komercyjnie.

Odczytywanie konfiguracji aplikacji wymaga obsługi wielu formatów plików. Proponowane rozwiązanie obsługuje jedynie pliki XML i język tworzenia zapytań XPath. Aby osiągnąć wyższą jakość rozwiązania, niezbędna jest obsługa także innych formatów.

Utrzymanie Bazy Wiedzy w formie arkusza kalkulacyjnego nie jest dobrym wyborem długoterminowym. W przypadku dostępu i modyfikacji arkusza przez wielu użytkowników, można spodziewać się wielu konfliktów. Ten obszar systemu będzie w przyszłości musiał być zastąpiony lepszym rozwiązaniem.

6.4.2 Dalsze prace

Proponowane rozwiązanie jest zaledwie pierwszym krokiem w kierunku tworzenia automatycznego systemu konfiguracji aplikacji.

Istnieje możliwość całkowitej automatyzacji procesu konfiguracji aplikacji uruchomionych w chmurze. Takie podejście wymaga jednak pewnego okresu treningu modelu Uczenia Maszynowego. Planowane są w tym celu dalsze badania, które będą prowadzone w ramach laboratorium firmy Red Hat.

Rozdział 7

Podsumowanie pracy i uwagi końcowe

Poniżej przedstawiono najważniejsze rezultaty pracy, ze szczególnym uwzględnieniem rezultatów o charakterze nowatorskim oraz wynikające z nich wnioski:

- Teza niniejszej pracy została pozytywnie zweryfikowana. Przedstawione rozwiązania udowadniają, że możliwe jest poprawienie szybkości komunikacji, zdefiniowanej jako przepustowość lub opóźnienie, oraz obniżenie zapotrzebowania na pamięć poprzez wykorzystanie nowych rozwiązań i algorytmów negocjacji protokołów oraz rozwiązań wykorzystujących technikę ”Client Side Load Balancing” dla aplikacji uruchomionych w chmurze.
- Zaproponowano i opracowano następujące rozwiązania:
  - Obniżenie zapotrzebowania na pamięć systemu typu Data Grid przez umożliwienie współdzielenie serwera pomiędzy wielu użytkowników. Zaproponowana metoda wykorzystuje pole “hostname” rozszerzenie SNI protokołu TLS oraz uwierzytelnianie za pomocą certyfikatu X509. Rozwiązanie znalazło zastosowanie praktyczne w projekcie Infinispan.
  - Przepustowość komunikacji została zwiększona dzięki zastosowaniu techniki “Client Side Load Balancing” oraz dzięki proponowanemu rozwiązaniu dostępu do aplikacji uruchomionej w chmurze. Pomysł wykorzystany w rozwiązaniu znalazł zastosowanie w implementacji złożonych funkcjonalności systemu Data Grid, takich, jak replikacja danych pomiędzy klastrami.

- Przepustowość połączenia pomiędzy serwerem i aplikacją kliencką zostało zwiększone poprzez zastosowanie zaproponowanej techniki zmiany protokołów z wykorzystaniem rozszerzenie ALPN protokołu TLS oraz procedury HTTP/1.1 Upgrade i wykorzystanie binarnych protokołów komunikacyjnych. Rozwiązanie znalazło praktyczne zastosowanie i jest obecnie domyślną konfiguracją w wersji 10 projektu Infinispan.
- Zaproponowano System Ekspercki do wykrywania błędów w konfiguracji aplikacji uruchomionych w chmurze. System został pozytywnie zweryfikowany przez osoby utrzymujące projekty OpenSource oraz przez obsługę techniczną firmy Red Hat. Zaproponowane rozwiązanie zostało zgłoszone w Biurze Patentowym Stanów Zjednoczonych.

Wiele rozwiązań prezentowanych w niniejszej pracy zostało zastosowanych w projektach OpenSource takich, jak JGroups, czy Infinispan. Sprawia to, że proponowane rozwiązania rozwiązują praktyczne problemy charakterystyczne dla systemów typu "Data Grid" i chmur opartych o kontenery.