POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Telekomunikacja
SPECJALNOŚĆ: Teleinformatyka i multimedia

PRACA DYPLOMOWA MAGISTERSKA

Analiza możliwości i implementacja niezawodnego routingu w wirtualnym środowisku High Avability przy wykorzystaniu platformy Kubernetes oraz Calico.

Analysis of possibilities and implementation of reliable routing in the virtual High Avability environment using the Kubernetes and Calico platforms.

AUTOR:
Rafał Dziwiński

PROWADZĄCY PRACĘ:
dr inż. Waldemar Grzebyk, KTT

OCENA PRACY:

WROCŁAW 2019

Spis treści

Rozdział 1

Wstęp

W dzisiejszych czasach... blablabla (wstęp i cel)

Rozdział 2

Cel i zakres pracy

Aspekt badawczy: 1. Opracowanie zakresu analizy oraz kryteriów oceny. 2. Analiza możliwości implementacji niezawodnego routingu w wirtualnym środowisku High Avability przy wykorzystaniu platformy Kubernetes oraz Calico: a) analiza sposobów implementacji niezawodnego routingu w środowisku wirtualnym, b) porównanie rozwiązania zwirtualizowanego do fizycznego w oparciu o przyjęte kryteria. Aspekt inżynierski: 1. Przygotowanie środowiska testowego wykorzystującego platformę Kubernetes oraz implementacja wirtualnych routerów jako kontenerów przy pomocy aplikacji Docker. 2. Przygotowanie scenariuszy i procedur testowych. 3. Przeprowadzenie testów dotyczących czasu odzyskania pełnej funkcjonalności sieci w momencie awarii jednego bądź wielu z jej elementów. Zadania do wykonania: 1. Studia literaturowe. 2. Opracowanie zakresu analizy oraz kryteriów oceny. 3. Analiza możliwości implementacji niezawodnego routingu w wirtualnym środowisku High Avability przy wykorzystaniu platformy Kubernetes oraz Calico: a) analiza sposobów implementacji niezawodnego routingu w środowisku wirtualnym, b) porównanie rozwiązania zwirtualizowanego do fizycznego w oparciu o przyjęte kryteria. 4. Przygotowanie środowiska testowego wykorzystującego platformę Kubernetes oraz implementacja wirtualnych routerów jako kontenerów przy pomocy aplikacji Docker. 5. Przygotowanie scenariuszy i procedur testowych. 6. Przeprowadzenie testów dotyczących czasu odzyskania pełnej funkcjonalności sieci w momencie awarii jednego bądź wielu z jej elementów. 7. Opracowanie wniosków z przeprowadzonej analizy i testów.

Rozdział 3

Część teorytyczna

BlaBlaBla

3.1 Słownik pojęć

Pod najmniejsza możliwa do wdrożenia jednostka obliczeniowa zarządzana przez platformę Kubernetes.

Node maszyna robocza zawierająca usługi niezbędne do uruchamiania podów oraz zarządzania nimi.

Master nod maszyna robocza zapewniająca płaszczyznę sterowania klastra.

Worker node maszyna robocza zapewniająca zasoby sprzętowe dla tworzonych podów.

Host endpoint reprezentuje jeden bądź wiele fizycznych bądź wirtualnych interfjesów połączonych z hostem na którym uruchomione jest calico.

Workload endpoints reprezentuje zasób wirtualny podłączony do sieci Calico.

Control plane descriptioncontrol plane aadsaef

3.2 Konteneryzacja

Konteneryzacja Aby opisać w dobry sposób konteneryzację warto przypomnieć sobie jak było kiedyś. Pierwszą metodą wdrażania aplikacji było uruchomienie każdej z nich na osobnym fizycznym serwerze. Niestety, w tym przypadku

nie ma możliwości wydzielenia zasobów dla danej aplikacji co powoduje problemy z jednoczesnym uruchomieniem wielu aplikacji. Może się zdażyć tak, że duże obciążenie na przykład przez atak DDoS spowoduje zatrzymanie pracy innych, zupełnie niezwiązanych ze sobą usług. Jedynym wyjściem jest uruchomienie aplikacji na osobnych fizycznych maszynach jednak generuje to szereg problemów. Między innymi:

- Trudność w skalowaniu zasobów;
- zwiększenie ilości niewykorzystanych zasobów;
- duże koszty.

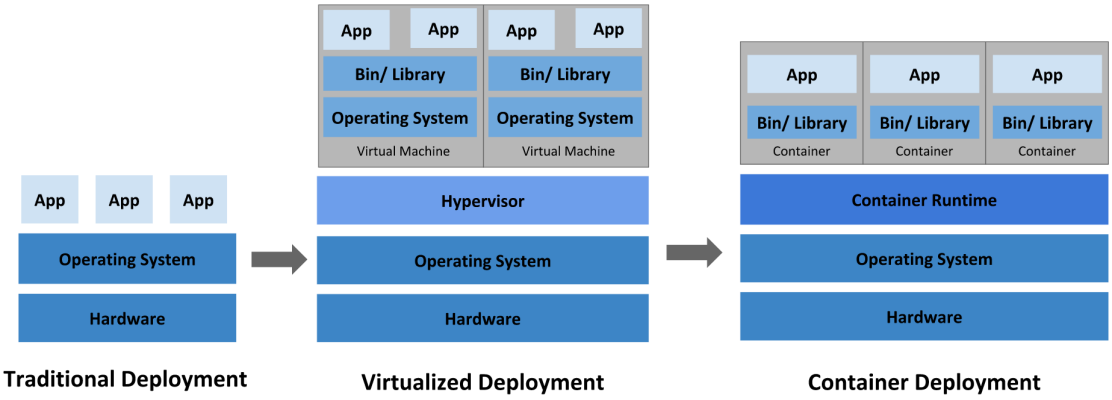
Aby temu zaradzić wprowadzono pojęcie wirtualizacji. Pozwala ona na uruchomienie wielu wirtualnych maszyn (VM) na jednym fizycznym serwerze. Pozwala to na izolację aplikacji co ma wpływ na stabilność w przypadku przeciążenia jednej z aplikacji oraz bezpieczeństwo. Kolejnym plusem jest możliwość lepszego zarządzania oraz skalowania zasobów sprzętowych. Zmniejsza również koszt wdrożenia oraz utrzymania dzięki możliwości zastosowania jednego, wydajnego serwera oraz ograniczenie kosztu prądu. W tym przypadku każda z VM wirtualizuje fizyczne zasoby oraz sprzęt co również nie jest rozwiązaniem w stu procentach wydajnym.

Na pomoc przychodzi tak zwana konteneryzacja. Jest to rozwiązanie podobne do maszyn wirtualnych z tą różnica, że nie jest potrzebna wirtualizacja sprzętu oraz systemu operacyjnego. Kontener działa na jądrze systemu hosta oraz jego zasobach które nie są wirtualizowane. Oznacza to, że w przypadku wdrożenia aplikacji w kontenerze potrzebne jest jedynie dodanie do niego pakietów charakterystycznych dla tej aplikacji. Na przykład w momencie uruchomienia dystrybucji Fedora na fizycznym serwerze z Debianem kontener będzie zawierał jedynie pakiety charakterystyczne dla tej pierwszej dystrybucji. W ogromnych skrócie można powiedzieć, że jest to lżejsza wersja VM. Ponadto użycie kontenerów niesie za sobą wiele zalet:

- Większa łatwość i wydajność tworzenia obrazu kontenera w porównaniu do użycia obrazu VM.
- Łatwiejszy rozwój aplikacji przez możliwość szybkiego budowania kolejnych obrazów.
- Możliwość łatwiej migracji między wieloma różnymi usługami w chmurze.
- Centralne monitorowanie oraz zarządzanie konternerami.

- Wydajniejsze oraz efektywniejsze zarządzanie zasobami sprzętowymi.

Różnicę między tradycyjnym wdrażaniem aplikacji a VM oraz kontenerami najlepiej obrazuje i podsumowuje poniższa grafika:



Rysunek 3.1 Ewolucja możliwości wdrażania aplikacji [1].

3.3 Kubernetes

Słowo Kubernetes wywodzi się z języka greckiego i oznacza sternika bądź pilota. Pierwsze linie kodu napisane przez Google Inc. pojawiły się już w 2004 roku. Platforma bazuje na wieloletnim doświadczeniu tej firmy oraz społeczności.

Jest to otwarta platforma służąca do zarządzania skonteneryzowanymi aplikacjami oraz usługami. Głównym jej celem jest ułatwienie i automatyzacja ich wdrożeń oraz konfiguracji. Obecnie społeczność oraz firmy stojące za rozwojem systemu są na tyle liczne, że z powodzeniem ta młoda platforma może być wdrażana w środowiskach produkcyjnych.

Jak wspomniano w poprzednim rozdziale najlepszym z rozwiązań do uruchamiania aplikacji są kontenery. W środowisku produkcyjnym, liczącym wiele dziesiątek bądź też setek kontenerów ręczne zarządzanie nimi jest niewydajne i obarczone ryzykiem wielu błędów. Również w momencie awarii konieczna jest szybka reakcja na przykład poprzez utworzenie zastępczych kontenerów. W tym celu bardzo dobrym wyjściem jest zastosowanie tak zwanego orkiestratora jakim jest Kubernetes. Do wymienionych wyżej zalet możemy dołączyć:

- Wykrywanie usług oraz load balancing;

- automatyczne montowanie wybranych typów pamięci takich jak pamięć lokalna bądź zasób sieciowy; automatyczne przydzielanie zasobów sprzętowych dla każdego z kontenerów;
- możliwość samoczynnego restartowania kontenera w momencie wykrycia awarii;
- łatwe zarządzanie danymi poufnymi wykorzystywanymi w kontenerach oraz zarządzanie ich konfiguracją.

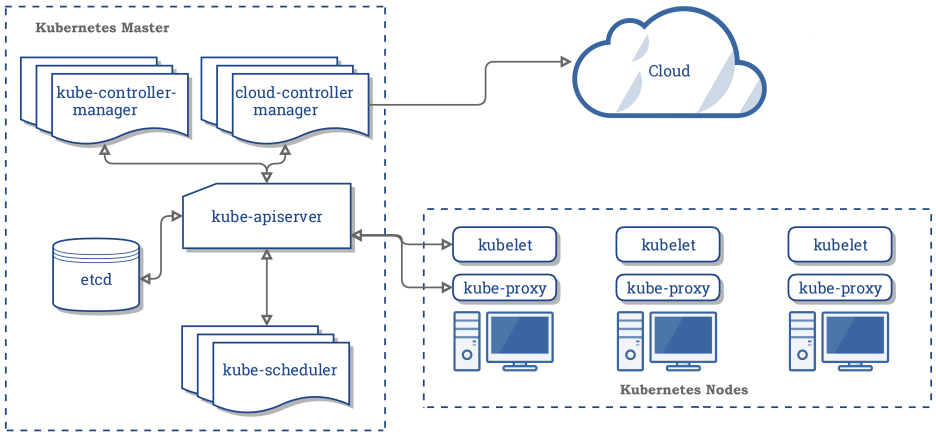
3.3.1 Komponenty

W momencie wdrażania systemu Kubernetes nie uruchamiana jest pojedyncza aplikacja a szereg połączonych ze sobą usług. Aby to zrobić potrzebny jest tak zwany klaster składający się z minimum dwóch węzłów (node):

- Worker node na której uruchamiane są kontenery
- Master node zarządzający worker nodami oraz kontenerami w klastrze.

Przykładowy klaster wraz z wyszczególnionymi nodami został zaprezentowany poniżej.

]



Rysunek 3.2 Przykładowy klaster wraz z komponentami [1].

3.3.1.1 Master Node

Jak już wspomniano master node służy do zarządzania klastrem oraz kontenerami. Podejmuje on decyzje o umieszczeniu danego poda w node oraz zarządza ich zasobami. Składa się z szeregu komponentów które zostały wypisane oraz opisane poniżej:

kube-apiserver komponent control plane kubernetesa odpowiedzialny za komunikację użytkownika z platformą.

etcd silnie spójny i wysoce dostępny, rozproszony magazyn w formie klucz-wartość. Zapewnia niezawodny sposób przechowywaniakonfiguracji kubernetesa. Jeden z ważniejszych komponentów któremu konieczne jest zapewnienie nadmiarowości. Aby móc tolerować awarię jednego z modułów konieczne jest stworzenie klastra z ilością trzech.

kube-scheduler komponent który w momencie wykrycia utworzenia nowego poda przypisuje go do odpowiedniego node bądź wielu. Decyzje te podejmuje na podstawie wymagań dotyczących zasobów/sieci poda oraz obciążenia obecnych workerów w klastrze.

kube-controller-manager komponent służący do uruchamiania kontrolerów. (https://kubernetes.
Każdy z kontrolerów jest osobnym procesem, lecz aby zwiększyć wydajnośc kompilowane są do jednego pliku binarnego. Zarządza on następującymi kontrolerami:

node controller odpowiada za rekacje w momencie awarii jednego z node.

replication controller zapewnia nadmiarowość podów w klastrze.

endpoints controller zapewnia spójność usług oraz podów

service account and token controllers tworzy użytkowników oraz zapewnia dostęp do API.

cloud-controller-manager zapewnia integrację z siecią dostawcy usług chmurowych.

3.3.1.2 Node

Poza kluczowymi komponentami wymienionymi w rozdziale wyżej które występują tylko w master node istnieją również dwa uruchamiane także na worker node.

- kubelet - komponent którego zadaniem jest uruchamianie kontenerów wewnątrz podów.
- kube-proxy - zarządza regułami sieciowymi w node pozwalającymi na komunikację podów z zasobami wewnątrz oraz poza klastrem. Jeżeli jest taka możliwość używa on zasoby oraz mechanizmy filtracji pakietów hosta.
- Container Runtime - komponent uruchamiający kontenery w klastrze. Na przykład: Docker, Containerd cri-o, bądź inny wspierany.

3.4 Calico

Calico jest to oprogramowanie wykorzystywane do zapewnienia sieci oraz jej bezpieczeństwa w środowiskach zwirtualizowanych oraz skonteneryzowanych. Jest ono otwarte i może być wykorzystywane przez wiele systemów takich jak OpenStack, OpenShift, Docker EE oraz Kubernetes. Głównymi założeniami twórców było stworzenie rozwiązania skalowalnego w środowiskach opartych o rozwiązania chmurowe oraz zapewnienie wydajności zbliżonej do rozwiązań oparty o jądro linuxa. Największymi zaletami Calico są:

- Bezpieczeństwo - Użycie modelu "zero trust network" - wszystko co nie jest dozwolone jest zakazane.
- Wydajność - Calico korzysta z wbudowanych w jądro linuxa, wysoko zoptymalizowanych rozwiązań przekazywania oraz filtracji pakietów. W większości przypadków zbędne jest użycie funkcji enkapsulujących oraz dekapsulujących.
- Skalowalność - Osiągnięto ją dzięki wykorzystaniu najlepszych wzorców projektowych dla środowisk zwirtualizowanych oraz najpopularniejszych i najlepiej sprawdzonych protokołów sieciowych. Calico w cyklu testów deweloperskich jest uruchamiane na klastrze posiadającym wiele tysięcy node's.
- Uniwersalność - Poza zapewnieniem sieci dla systemów takich jak Kubernetes możliwe jest również wykorzystywanie w klasycznych maszynach wirtualnych a także interfejsach fizycznych.

3.4.1 Host endpoints

Calico poza możliwością definiowania punktów końcowych dla podów, maszyn wirtualnych bądź kontenerów umożliwia również tworzenie endpoints dla fizycznych interfejsów. Tak samo host endpoints może zawierać etykiety które znajdują się w tej samej przestrzeni nazw co workload endpoints. Calico nie wspiera konfiguracji adresów IP bądź konfiguracji w oparciu na adresach MAC interfejsów, zatem konieczna jest ich konfiguracja przez sieć podkładową naszego systemu.

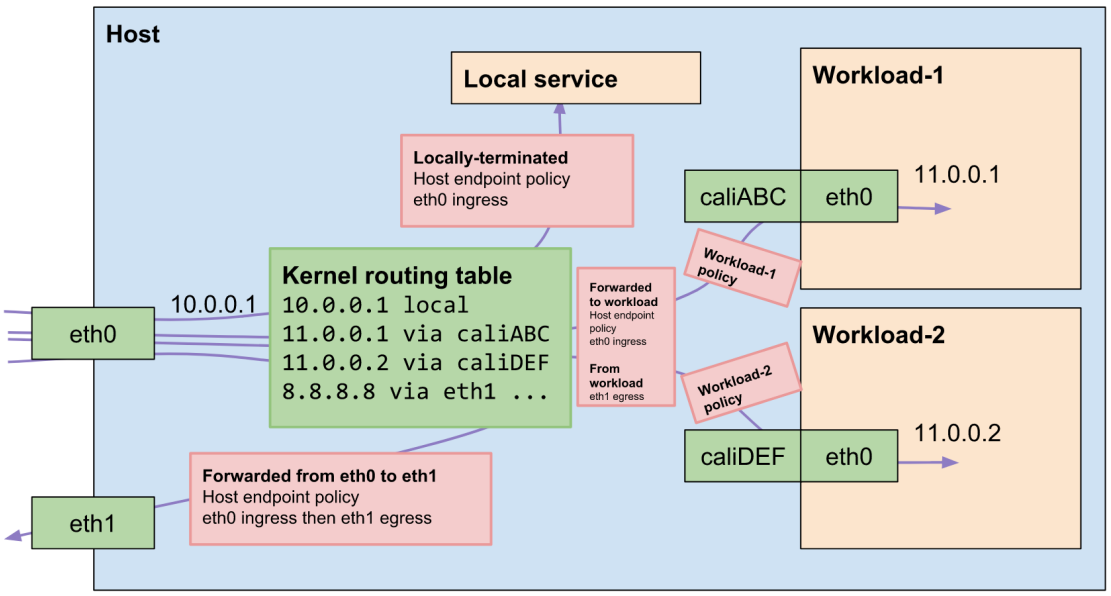
Calico rozróżnia workload endpoints od host endpoints dzięki konfigurowalnym prefiksom. Dzięki polu InterfacePrefix w konfiguracji modułu Felix możemy zdecydować, że punkty końcowe zaczynające się od danej wartości będą traktowane jako interfejsy workload. Pozostałe natomiast oznaczać będą interfejsy hosta.

Calico domyślnie blokuje cały ruch między workload interfaces, pozwalając tylko na ruch gdy interfejs jest znany oraz skonfigurowane są reguły bezpieczeństwa. W przypadku host endpoints Calico jest bardziej pobłażliwe. Monitoruje tylko ruch między zdefiniowanymi interfejsami, natomiast nie analizuje wszelkiego innego.

Możemy zastosować reguły bezpieczeństwa w kontekście punktów końcowych hosta dla trzech rodzajów ruchu:

- zakończonego lokalnie,
- przekazywanego między host endpoints
- przekazywanego pomiędzy host endpoint a workload endpoint.

Przykład zastosowania wszystkich trzech rodzajów ruchu zaprezentowano na grafice poniżej;



Rysunek 3.3 Schmat pokazujący działanie calico w kontekście host endpoints [2].

Rozdział 4

Bibliografia

[1] Dokumentacja projektu Kubernetes,
https://kubernetes.io/docs/home/

[2] Dokumentacja projektu Calico
https://docs.projectcalico.org/v3.10/introduction/

[3] Kubernetes in Action, Manning Publications 2017
ISBN: 9781617293726

[4] Kubernetes: Up and Running, O'Reilly Media, Inc. 2017
ISBN: 9781491935675

[5] Learn Docker - Fundamentals of Docker 18.x, Packt Publishing 2018
ISBN: 9781788997027