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 2020

Spis treści

Rozdział 1

Wstęp

Rozdział 2

Cel i zakres pracy

Rozdział 3

Część teoretyczna

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, wirtualnych interfejsów połączonych z hostem, na którym uruchomione jest calico.

Workload endpoints reprezentuje zasób wirtualny, podłączony do sieci Calico.

Control plane płaszczyzna sterowania zapewniająca logikę działania systemu.

3.2 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. Powoduje to problemy z jednoczesnym uruchomieniem wielu aplikacji. Może się zdarzyć 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:

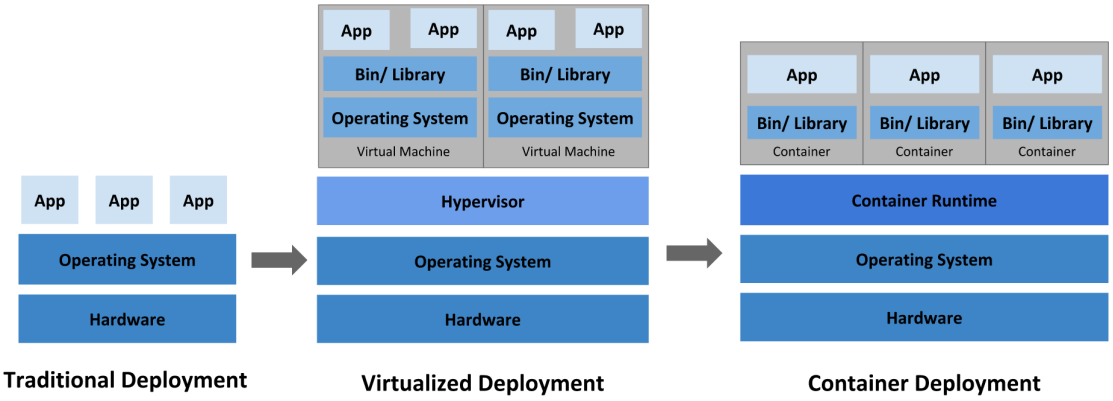
- Trudność w skalowaniu zasobów;
- zwiększenie ilości niewykorzystanych zasobów;
- duże koszty.

Wprowadzono pojęcie wirtualizacji, aby temu zaradzić. Pozwala ona na uruchomienie wielu wirtualnych maszyn (VM), na jednym fizycznym serwerze. Dzięki temu możliwa jest separacja 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 przez nią wymaganych. Na przykład, w momencie uruchomienia dystrybucji Fedora na fizycznym serwerze z Debianem kontener będzie zawierał jedynie pakiety charakterystyczne dla tej pierwszej. W ogromnych skrócie można powiedzieć, że jest to lżejsza wersja VM. Ponadto użycie ich użycie 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 administracja kontenerami.
- 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 bądź konfiguracji. Obecnie społeczność, a także 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 ich wiele dziesiątek bądź też setek, 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 dysk twardy 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.

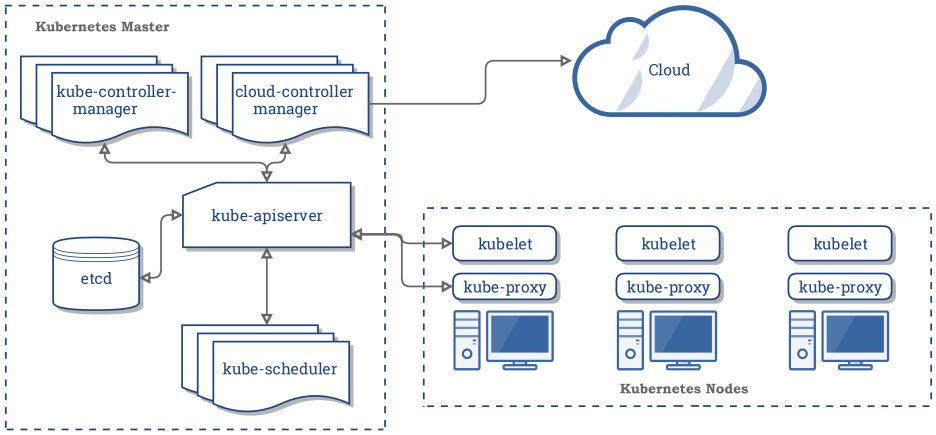
3.3.1 Komponenty

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 przechowywania konfiguracji 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 wydajność kompilowane są do jednego pliku binarnego. Zarządza on następującymi kontrolerami:

node controller odpowiada za reakcje, 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 i skonteneryzowanych. Jest ono otwarte i może być wykorzystywane przez wiele systemów takich jak OpenStack, OpenShift, Docker EE oraz Kubernetes. Głównymi założeniami twórców było stworzenie systemu skalowalnego w środowiskach opartych o rozwiązania chmurowe. Ponadto, kluczowym aspektem było 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 znadują się w tej samej przestrzeni nazw co workload endpoints. Calico nie wspiera konfiguracji adresów IP, bądź konfiguracji w oparciu na adresach MAC interfejsów. Konieczna jest zatem ich konfiguracja przez sieć podkładową naszego systemu.

Calico rozróżnia workload endpoints od host endpoints, za pomocą konfigurowalnych prefiksów. Dzięki polu InterfacePrefix w konfiguracji modułu Felix możemy zdecydować, że punkty końcowe zaczynające się od danej wartości będą

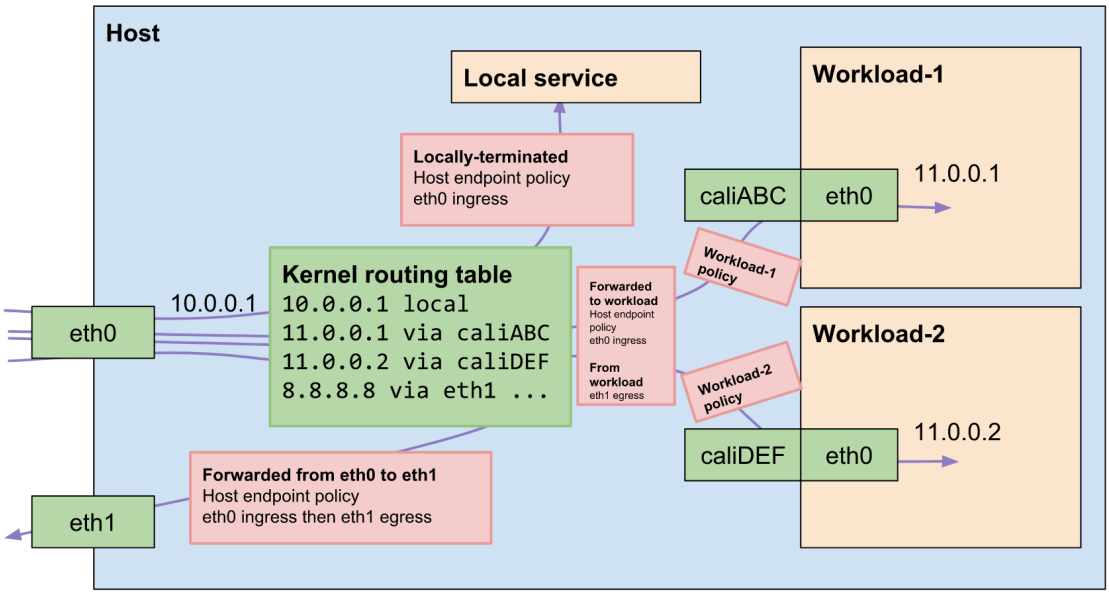
traktowane jako interfejsy workload. Pozostałe natomiast, będą oznaczać interfejsy hosta.

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

Uruchomienie klastra

W celu realizacji pracy magisterskiej, stworzono klaster Kubernetes'a składający się z 6 node. W celu symulacji maszyn fizycznych, wykorzystano narzędzie KVM. Każdy z node, to osobna maszyna wirtualna, połączona ze sobą przy pomocy sieci izolowanej. Dodatkowo, każdy z node posiada adres publiczny, w celu zapewnienia dostępu do internetu. Systemem operacyjnym dla node jest Ubuntu 18.04.3 LTS. Reasumując, wykorzystano maszyny wirtualne zgodnie z tabelą 4.

Nazwa hosta	Adres IP lokalny	Adres IP publiczny	CPU	RAM	Dysk
k8s-01-master	10.10.1.1	212.127.89.81	4 Core	4 GB	40GB
k8s-02-master	10.10.1.2	212.127.89.82	4 Core	4 GB	40GB
k8s-03-master	10.10.1.3	212.127.89.83	4 Core	4 GB	40GB
k8s-04-worker	10.10.1.4	212.127.89.84	2 Core	2 GB	40GB
k8s-05-worker	10.10.1.5	212.127.89.85	2 Core	2 GB	40GB
k8s-06-worker	10.10.1.6	212.127.89.86	2 Core	2 GB	40GB

Tablica 4.1 Lista node klastra

Wykorzystanie trzech master node wymusza użycie loadbalancera. Do tego celu wykorzystano kolejną maszynę wirtualną z zainstalowanym HAProxy. Konfiguracja ta pozwala na awarię jednego master node oraz dwóch worker node bez wpływu na działanie klastra.

W celu uruchomienia klastra wykorzystano narzędzie kubeadm wspierające nas w jego tworzeniu. Dzięki temu, uruchomienie Kubernetesa staje się szybsze bez uszczerbku na wydajności. Dużym atutem tego narzędzia, jest możliwość bardzo szybkiego dodania kolejnego worker node w razie potrzeby. Kolejne kluczowe kroki, wraz z komendami zostały opisane poniżej.

4.1 Loadbalancer

Jako loadbalancer użyto narzędzia HAProxy zainstalowanego na maszynie wirtualnej o poniższych parametrach: Pakiet haproxy znajduje się w repozytorium,

Nazwa hosta	Adres IP lokalny	Adres IP publiczny	CPU	RAM	Dysk
k8s-loadbalancer	10.10.1.100	Brak	2 Core	2 GB	40GB

Tablica 4.2 Parametry loadbalancer'a.

zatem:

```
sudo apt install haproxy
```

Następnie należy skonfigurować loadbalancer za pomocą pliku *haproxy.cfg* w katalogu */etc/haproxy/*. Zawartość pliku konfiguracyjnego znajduje się poniżej.

```
frontend kubernetes
bind 10.10.1.100:6443
option tcplog
mode tcp
default_backend kubernetes-master

backend kubernetes-master
mode tcp
balance roundrobin
option tcp-check
server k8s-01-master 10.10.1.1:6443 check fall 3 rise 2
server k8s-02-master 10.10.1.2:6443 check fall 3 rise 2
server k8s-03-master 10.10.1.3:6443 check fall 3 rise 2
```

4.2 Wyłączenie nftables

Kubernetes wykorzystuje iptables i jest przygotowany do pracy z tym modułem. Najnowsze dystrybucje takie jak Debian Buster, Ubuntu 19.04, Fedora 29 wykorzystują nowsze narzędzie nftables. Należy pamiętać, aby iptables nie wykorzystywało nftables jako backend. W przypadku wykorzystania Ubuntu 18.04.03 LTS można ten krok pominąć, jednak warto mieć to na uwadze.

```
sudo -i
update-alternatives --set iptables /usr/sbin/iptables-legacy
update-alternatives --set ip6tables /usr/sbin/ip6tables-legacy
```

```
update-alternatives --set arptables /usr/sbin/arptables-legacy
update-alternatives --set ebtables /usr/sbin/ebtables-legacy
```

4.3 Aktualizacja systemu oraz instalacja niezbędnych paczek

Przed instalacją pakietów, konieczna jest aktualizacja systemu oraz dodanie repozytorium Kubernetesa.

```
sudo -i
apt-get update && apt upgrade -y
apt-get install -y apt-transport-https ca-certificates curl software-properties-
    common
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
cat <<EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF
apt-get update
```

Następnie należy zainstalować system kontenerów, czyli Docker. Wersja Kubernetesa 1.17 wymaga go w maksymalnie wersji 19.03 i taka też zostanie zainstalowana. Warto pamiętać, o blokadzie automatycznej aktualizacji w celu zapewnienia stabilności klastra.

```
sudo -i
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -
cat <<EOF | sudo tee /etc/apt/sources.list.d/docker.list
deb https://download.docker.com/linux/ubuntu bionic stable
EOF
apt-get update && apt-get install -y \
docker-ce=5:19.03.4~3-0~ubuntu-$(lsb_release -cs) \
docker-ce-cli=5:19.03.4~3-0~ubuntu-$(lsb_release -cs)
apt-mark hold docker-ce docker-ce-cli
```

Po instalacji Docker'a należy zmienić domyślny sterownik cgroup z cgroupfs, na rekomendowany systemd.

```
sudo -i
cat > /etc/docker/daemon.json <<EOF
{
"exec-opts": ["native.cgroupdriver=systemd"],
"log-driver": "json-file",
"log-opts": {
"max-size": "100m"
},
```

```
"storage-driver": "overlay2"
}
EOF

mkdir -p /etc/systemd/system/docker.service.d

systemctl daemon-reload
systemctl restart docker
```

Ostatnim krokiem jest instalacja kubeadm oraz pakietów Kubernetesa. Przed jego uruchomieniem, konieczne jest wyłączenie partycji SWAP.

```
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

4.4 Stworzenie master oraz worker node

Jak już wspomniano, do instalacji klastra wykorzystano narzędzie kubeadm, pozwalające przyśpieszyć wiele procesów instalacyjnych. Najważniejszymi poleceniami wspomnianej aplikacji, jest kubeadm init oraz kubeadm join. Najpierw zostało opisane polecenie inicjalizujące control plane klastra tj. kubeadm init. Posiada ono, szereg argumentów pozwalających na dostosowanie klastra do własnych potrzeb. Najważniejsze opisano poniżej:

control-plane-endpoint Przydatne szczególnie, w komecie rozbudowy klastra o kolejny master node. Umożliwia ustawienie wspólnego punktu końcowego, dla wszystkich węzłów control plane.

pod-network-cidr Określa pulę adresów IP, wykorzystywanych dla sieci z podami.

apiserver-advertise-address Domyślnie kubeadm używa interfejsu przypisane do bramy domyślnej jako adresu docelowego serwera API master node.

upload-certs Opcja ta pozwala na wysłanie potrzebnych certyfikatów do bazy kubeadm.

Zatem, polecenie inicjalizujące pierwszy control plane klastra to:

```
kubeadm init --control-plane-endpoint 10.10.1.100 --pod-network-cidr
    192.168.0.0/18 --apiserver-advertise-address 10.10.1.100 --upload-certs
```

Jeżeli wszystko zostanie sprawnie zainstalowane, będziemy o tym poinformowani stosownym komunikatem. Zawiera on również informację o możliwości dodania kolejnego node. Jego fragment został zaprezentowany poniżej:

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

You can now join any number of control-plane nodes by copying certificate
    authorities
and service account keys on each node and then running the following as root:

kubeadm join 10.10.1.100:6443 --token xx.xxx \
--discovery-token-ca-cert-hash sha256:xxxx \
--control-plane

Then you can join any number of worker nodes by running the following on each as
    root:

kubeadm join 10.10.1.100:6443 --token xx.xxx \
--discovery-token-ca-cert-hash sha256:xxxx
```

Dodajemy kolejno dwa master node oraz trzy worker node. Dzięki wykorzystaniu kubeadm możliwe jest to za pomocą jednego polecenia, odpowiednio z flagą *–control-plane* dla master node, bądź bez.

```
kubeadm join 10.10.1.100:6443 --token xx.xxx --discovery-token-ca-cert-hash sha256
    :xxxx
```

W przypadku braku tokenu, możliwe jest jego stworzenie wykonując poniższe polecenie na master node:

```
kubeadm token create
```

natomiast hash certyfikatu zostanie wyświetlony poleceniem:

```
openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -outform
    der 2>/dev/null \
| openssl dgst -sha256 -hex | sed 's/^.* //'
```

Gotowość klastra możemy zweryfikować poprzez wypisanie działających node.

```
rdziwinski@k8s-01-master:~$ kubectl get node
NAME            STATUS     ROLES    AGE     VERSION
k8s-01-master   Ready      master   7d21h   v1.17.2
k8s-02-master   Ready      master   7d23h   v1.17.2
k8s-03-master   Ready      master   7d23h   v1.17.2
k8s-04-worker   Ready      <none>   7d22h   v1.17.2
k8s-05-worker   Ready      <none>   7d22h   v1.17.2
k8s-06-worker   Ready      <none>   7d22h   v1.17.2
```

Master i worker node są gotowe do pracy. Kolejnym krokiem jest ręczne przypisanie adresu IP dla każdego z node. W tym celu, na każdym z node należy edytować plik konfiguracyjny dodając argument –node-ip=10.10.1.x do zmiennej KUBELET_CONFIG_ARGS:

```
vim /etc/systemd/system/kubelet.service.d/10-kubeadm.conf
```

oraz zresetować procesy

```
systemctl daemon-reload
systemctl restart kubelet
```

4.5 Calico

Jak już wspomniano, do zapewnienia sieci wewnątrz klastra wykorzystano narzędzie Calico. Instalacja jego odbywa kolejno w trzech krokach:

1. Pobranie pliku konfiguracyjnego ze strony:
   https://docs.projectcalico.org/v3.11/manifests/calico.yaml
2. Edycja wartości *CALICO_IPV4POOL_CIDR* zgodnie z zadeklarowaną wcześniej pulą.
3. Wykonanie pliku konfiguracyjnego. *kubectl apply -f calico.yaml*

Poprawną instalację możemy zweryfikować poprzez sprawdzenie obecności podów wykorzystywanych przez system calico. Jeżeli wszystko jest poprawne powinniśmy zobaczyć informacje jak poniżej:

```
rdziwinski@k8s-01-master:~$ kubectl get pods -n kube-system
NAME                                       READY   STATUS    RESTARTS   AGE
calico-kube-controllers-5b644bc49c-zdhqn   1/1     Running   227        3d21h
calico-node-5hcfb                          1/1     Running   4          3d21h
calico-node-8d7h2                          1/1     Running   9          3d21h
calico-node-dwrjp                          1/1     Running   9          3d21h
```

```
calico-node-ghtkc                              1/1     Running   4          3d21h
calico-node-mtkkx                              1/1     Running   4          3d21h
calico-node-nqpgl                              1/1     Running   9          3d21h
coredns-6955765f44-2r7ch                       1/1     Running   5          3d21h
coredns-6955765f44-gm2b5                       1/1     Running   3          3d22h
etcd-k8s-01-master                             1/1     Running   28         7d22h
etcd-k8s-02-master                             1/1     Running   39         7d23h
etcd-k8s-03-master                             1/1     Running   29         7d23h
kube-apiserver-k8s-01-master                   1/1     Running   55         7d22h
kube-apiserver-k8s-02-master                   1/1     Running   45         7d23h
kube-apiserver-k8s-03-master                   1/1     Running   45         7d23h
kube-controller-manager-k8s-01-master          1/1     Running   39         7d22h
kube-controller-manager-k8s-02-master          1/1     Running   43         7d23h
kube-controller-manager-k8s-03-master          1/1     Running   33         7d23h
kube-proxy-5sxq5                               1/1     Running   9          7d22h
kube-proxy-7mv2f                               1/1     Running   10         7d23h
kube-proxy-c26tt                               1/1     Running   9          7d22h
kube-proxy-gn72z                               1/1     Running   11         7d23h
kube-proxy-hkfcb                               1/1     Running   9          7d23h
kube-proxy-mm5lh                               1/1     Running   14         7d23h
kube-scheduler-k8s-01-master                   1/1     Running   44         7d22h
kube-scheduler-k8s-02-master                   1/1     Running   36         7d23h
kube-scheduler-k8s-03-master                   1/1     Running   33         7d23h
```

Aby móc pracować z calico konieczne jest zainstalowanie narzędzia calicoctl. Aby to zrobić należy pobrać plik binarny z repozytorium calico w serwisie github oaz skopiować do właściwego katalogu.

1. curl -O -L https://github.com/projectcalico/calicoctl/releases/download/v3.11.2/calicoctl
2. chmod +x calicoctl
3. sudo cp calicoctl /usr/local/bin/

Ostatnim etapem jest wskazanie calicoctl lokalizacji etcd. Można to zrobić na wiele sposobów, na przykład dopisując poniższe linie do pliku */.bashrc*:

```
export KUBECONFIG=/home/rdziwinski/.kube/config
export DATASTORE_TYPE=kubernetes
```

Aby uniknąć błędów w czasie pracy z klastrem, gdzie każdy z node ma kilka interfejsów sieciowych należy na stałe ustalić adres IP dla każdego z nich. Aby to zrobić, stworzono plik konfiguracyjny *nodes.yaml* w którym powielono poniższą konfiguracje dla każdego z node zmieniając jedynie wartość *ipv4Address*.

```
apiVersion: projectcalico.org/v3
kind: Node
metadata:
  name: k8s-01-master
```

```
spec:
  bgp:
    ipv4Address: 10.10.1.1/24
```

Konfigurację zatwierdzamy poleceniem *calicoctl apply -f nodes.yaml*.

Rozdział 5

Założenia oraz ich realizacja.

W ramach pracy stworzono projekt sieci firmowej w jednym z oddziałów. Kierowano się następującymi założeniami:

1. Każda z placówek posiada 3 sieci wewnętrzne:
   - Dział techniczny
   - Pozostali pracownicy
   - DMZ
2. W sieci dla pracowników zostanie uruchomiony serwer DHCP.
3. Izolacja urządzeń pomiędzy siecią działu technicznego a innych pracowników. Administratorzy powinni posiadać dostęp do urządzeń pracowników.
4. W DMZ będą znajdować się serwery www. Dwa z nich jako pody a jeden postawiony będzie jako osobna maszyna wirtualna.
5. Stworzony zostanie pod z instancją routera bird, wykorzystany do wymiany prefixów. W ramach pracy przyjęto, że adresacja rozgłaszana przez nasza firmę to 172.16.0.0/14, natmiast otrzymywana to 172.20.0.0/14.

Sieć firmową zasymulowano przy pomocy maszyn wirtualnych KVM. Klaster składa się z trzech master node oraz trzech worker nodów. Dodatkowo wykorzystano maszyny wirtualne jako komputery pracowników działu, loadbalancer oraz router z aplikacją Bird. Każdy z node posiada minimum dwa interfejsy:

- enp1s0 - adres prywatny z puli 10.10.1.0/24 służący do zarządzania oraz komunikacji między nodami.

Nazwa sieci	Adresacja	Interfejs
DMZ	192.168.64.0/22	enp7s0
Sieć pracowników	192.168.68.0/22	enp8s0
Sieć Administratorów	192.168.72.0/22	enp9s0

Tablica 5.1 Sieci wewnętrzne w firmie.

- eth6s0 - adres publiczny używany jako wyjście na świat.

Dodatkowo wszystkie trzy worker node posiadają interfejsy służące do wymiany ruchu między siecią fizyczną firm.

5.1 Sieci wewnętrzne

Sieć firmową podzielono na 3 podsieci wewnętrzne.

5.1.1 HostEndpoints

Jak już wspomniano Calico wykorzystuje zasób hostendpoints aby zdefiniować punkty końcowe sieci fizycznej. W tym celu należy stworzyć osobny zasób dla każdego z interfejsu fizycznego node, wykorzystywanego do działania sieci fizycznej. Poniżej przedstawiono fragment pliku konfiguracyjnego dla interfejsu wykorzystywanego przez pracowników.

```
apiVersion: projectcalico.org/v3
kind: HostEndpoint
metadata:
  name: k8s-06-enp8s0
  labels:
    interface: enp8s0
    subnet: pracownicy
spec:
  interfaceName: enp8s0
  node: k8s-06-worker
```

Warto zwrócić uwagę na etykietę *subnet* dzięki której jesteśmy w stanie pogrupować interfejsy w zależności od przeznaczenia. Dla zakończeń sieci używanych w przypadku ruchu publicznego, ze względów bezpieczeństwa, oraz wygody konfiguracji użyto innych etykiet. Poniżej znajduje się treść jednego z trzech takich zasobów.

```
apiVersion: projectcalico.org/v3
kind: HostEndpoint
metadata:
  name: k8s-04-enp6s0
  labels:
    interface: public
spec:
  interfaceName: enp6s0
  node: k8s-04-worker
```

5.1.2 Firewall

Jako narzędzie do separacji ruchu użyto mechanizmu Calico nazwanego GlobalNetworkPolicy. Konfiguracja różni się znacząco od tworzenia reguł w jednym z popularnych narzędzi typu firewall, jednak funkcja jego jest identyczna. Zgodnie z najlepszymi praktykami tworzenia wszelkich reguł bezpieczeństwa zaczęto od blokady każdego ruchu w klastrze.

```
apiVersion: projectcalico.org/v3
kind: GlobalNetworkPolicy
metadata:
  name: deny-all
spec:
  order: 999
  selector: has(subnet) #1
  applyOnForward: true #2
  types:
  - Ingress
  - Egress
  ingress:
  - action: Deny
  egress:
  - action: Deny
```

Kluczowe są dwa parametry:

1. linijka ta oznacza, że reguła dotyczy tylko obiektów które posiadają parametr *subnet*, tak jak stworzone wcześniej *hostendpoints*. Takie rozwiązanie zostało wdrożone aby nie zakłócić pracy innych elementów klastra.

2. dzięki ustawieniu zmiennej applyOnForward na true możliwe jest analizowanie ruchu który przechodzi przez wskazany *hostendpoints*. W przypadku pozostawienia tej wartości jako domyślnej Calico zastosowałoby tworzoną regułę tylko dla ruchu wychodzącego bądź docelowego do tego interfejsu.

W dalszej części pliku konfiguracyjnego blokowany jest ruch przychodzący oraz

wychodzący dla odpowiednich zakończeń sieci.

Aby umożliwić łatwiejsze zarządzanie adresacją w sieci stworzone zostały obiekt GlobalNetworkSet. Chcąc porównać do standardowych narzędzi tylu firewall, można powiedzieć, że jest to lista adresów. Fragment pliku konfiguracyjnego został zaprezentowany poniżej:

```
apiVersion: projectcalico.org/v3
kind: GlobalNetworkSet
metadata:
  name: administratorzy
  labels:
    pool: administratorzy
spec:
  nets:
  - 192.168.72.0/22
```

Jednym z założeń sieci firmowej, pokazujące możliwości Calico, było umożliwienie dostępu z sieci administratorów do sieci pozostałych pracowników. W tym celu stworzono kolejny plik konfiguracyjny wraz z regułą *GlobalNetworkPolicy* której którego zawartość znajduje się poniżej:

```
apiVersion: projectcalico.org/v3
kind: GlobalNetworkPolicy
metadata:
  name: allow-admins
spec:
  order: 4
  selector: subnet in {'administratorzy','pracownicy'} #1
  applyOnForward: true
  types:
  - Ingress
  - Egress
  ingress:
  - action: Allow #2
    protocol: ICMP
    source:
      selector: pool == 'administratorzy'
    destination:
      selector: pool == 'pracownicy'

  - action: Allow #3
    protocol: TCP
    source:
      selector: pool == 'administratorzy'
    destination:
      selector: pool == 'pracownicy'
    ports:
    - [55222,22]
```

```
    egress:
   - action: Allow
```

Warto zwrócić uwagę na wyszczególnione linie konfiguracji.

1. W tej linii następuje wybór interfejsów na które tworzona reguła będzie wpływać. Zatem wybieramy interfejsy do których podłączone będą urządzenia sieci dla administratorów oraz pozostałych pracowników.

2. Sekcja ta zezwala na ruch protokołu ICMP, gdzie źródłowy IP należy do sieci administratorzy a docelowy do sieci pracowników.

3. Analogicznie jak wyżej, jednak w tym przypadku wybieramy protokół TCP oraz porty 55222 i 22. Port 55222 jest portem ssh na którym wystawiony jest komputer w sieci pracowników.

5.1.3 Dostęp do internetu

Aby każda z sieci posiadała dostęp do internetu konieczne jest stworzenie zasobu Calico o nazwie ippool dla każdej z podsieci. Przykładowy plik konfiguracyjny dla sieci administratorów został zaprezentowany poniżej.

```
apiVersion: projectcalico.org/v3
kind: IPPool
metadata:
  name: administratorzy
  labels:
    subnet: administratorzy
spec:
  cidr: 192.168.72.0/22
  natOutgoing: true
```

Kluczowa w tej konfiguracji jest opcja *natOutgoing* której ustawienie na wartość *true* zezwala na natowanie adresu IP sieci lokalnej na adres publiczny.

Aby ruch wychodzący do internetu oraz przychodzący nie był blokowany należy dodatkowo utworzyć regułę na to zezwalającą. Jej treść znajduje się poniżej.

```
apiVersion: projectcalico.org/v3
kind: GlobalNetworkPolicy
metadata:
  name: internet
spec:
  order: 20
  selector: subnet in {'pracownicy','administratorzy'}
```

```
  applyOnForward: true
  types:
  - Ingress
  - Egress
  ingress:
  - action: Allow
    destination:
      notSelector: pool == 'local'
  egress:
  - action: Allow
    destination:
      selector: interface == 'public'
```

Działanie reguły bezpieczeństwa polega na umożliwieniu ruchu wychodzącego do *hostendpoints*, o nazwie internet, z publicznym adresem IP. Jednocześnie zezwalam na ruch przychodzący od sieci innych niż prywatne. Stworzenie takiej reguły zostało wsparte zasobem *globalnetworkset* którego treść znajduje się poniżej.

```
apiVersion: projectcalico.org/v3
kind: GlobalNetworkSet
metadata:
  name: local
  labels:
    pool: local
spec:
  nets:
  - 192.168.0.0/16
  - 172.16.0.0/12
  - 10.0.0.0/8
```

5.1.4 Blokada SSH

Możliwe jest również wykorzystanie Calico do ograniczenia dostępu bezpośrednio do klastra. Na przykład możemy zablokować logowanie przez ssh. W tym celu stworzono plik konfiguracyjny którego zawartość znajduje się poniżej:

```
apiVersion: projectcalico.org/v3
kind: GlobalNetworkPolicy
metadata:
name: cluster
spec:
  order: 16
  selector: interface == 'public'
  types:
  - Ingress
  - Egress
  ingress:
  - action: Allow
```

```
  - action: Deny
    protocol: TCP
    destination:
      ports: [55222,22]
  egress:
  - action: Allow
```

5.1.5 DHCP

W celu umożliwienia wygodnego korzystania z sieci, w podsieci dla pracowników uruchomiono serwer dhcp w postaci aplikacji *isc-dhcpd*. Zgodnie z założeniem uruchamiania usług wykorzystując kontenery i pody stworzono dwa pliki konfiguracyjne. Pierwszy z nich tworzy obiekt typu *DaemonSet*, natomiast drugi *ConfigMap*. Treść pierwszego z nich, wraz z komentarzem przedstawiono poniżej.

```
apiVersion: apps/v1
kind: DaemonSet #1
metadata:
  name: dhcpd
  labels:
    app: dhcpd
spec:
  selector:
    matchLabels:
      app: dhcpd
  updateStrategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: dhcpd
    spec:
      hostNetwork: true #2
      containers:
      - name: main
        image: docker.io/pnnlmiscscripts/dhcpd:4.4.1-4 #3
        imagePullPolicy: Always
        command: #4
        - /bin/sh
        - -ec
        - |
        touch /var/lib/dhcp/dhcpd.leases
        chown daemon.daemon /var/lib/dhcp/dhcpd.leases
        chown daemon.daemon /var/lib/dhcp
        chown daemon.daemon /var/run/dhcp
        dhcpd -cf /etc/dhcp/dhcpd.conf -d -user daemon -group daemon enp7s0 enp8s0
             enp9s0
      volumeMounts:
      - name: config
```

```
        mountPath: /etc/dhcp
    volumes:
    - name: config
      configMap: #5
        name: dhcpd
```

1. DaemonSet jest to specjalny rodzaj podu który tworzony jest na każdym z dostępnych worker node.

2. Konfiguracja ta umożliwia bezpośrednią widoczność serwera DHCP przez fizyczne interfejsy sieciowe node. jest to wymagane ze względu na mechanizm działania protokołu DHCP, który wymaga w standardowej konfiguracji widoczności w warstwie drugiej obu urządzeń.

3. Przed uruchomieniem aplikacji konieczna jest jej konfiguracja poprzez zmianę uprawnień dla używanych przez nią zasobów dyskowych. Ostatnia linijka bloku *command* uruchamia serwer dhcp dla wskazanych interfejsów.

4. Konfiguracja serwera nie jest zapisana na stałe w kontenerze a dynamicznie aktualizowana poprzez strukturę *configMap* opisaną w dalszej części.

Drugim z plików konfiguracyjnych jest obiekt *configMap* który umożliwia w prosty sposób wprowadzaniem zmian na wszystkich dostępnych podach na raz. Jego treść znajduje się poniżej.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: dhcpd
  labels:
    app: dhcpd
data:
  dhcpd.conf: |
    default-lease-time 600;
    max-lease-time 7200;
    ddns-update-style none;
    subnet 192.168.68.0 netmask 255.255.252.0 {
      range 192.168.68.10 192.168.68.240;
      option routers 192.168.71.254;
      option domain-name-servers 8.8.4.4;
      option subnet-mask 255.255.252.0;
    }
    subnet 192.168.72.0 netmask 255.255.252.0 {
      range 192.168.72.10 192.168.72.240;
      option routers 192.168.75.254;
      option domain-name-servers 8.8.4.4;
```

```
    option subnet-mask 255.255.252.0;
}
```

5.1.6 Brama domyślna

Jak można zauważyć, w konfiguracji serwera DHCP brama domyślna została zdefiniowana jako ostatni adres IP z każdej puli, na przykład: 192.168.75.254. Adresacja ta nie jest natomiast skonfigurowana na żadnym z interfejsów. Jest to spowodowane wykorzystaniem protokołu VRRP w celu zapewniania pełniej niezawodności sieci. Jako daemona wspomnianego protokołu wykorzystano aplikację *keepalived* .Konfiguracja jej polega na edycji pliku *keepalived.conf* znajdującego się w katalogu */etc/keepalived/*. Poniżej zamieszczono przykładową konfigurację dla node *k8s-05-worker*.

```
global_defs {
        vrrp_skip_check_adv_addr
}

vrrp_instance dmz {
        interface enp7s0
        state MASTER
        unicast_src_ip 192.168.67.245
        unicast_peer {
                192.168.67.244
                192.168.67.246
        }
        virtual_router_id 64
        priority 200
        virtual_ipaddress {
                192.168.67.254/22
        }
}

vrrp_instance pracownicy {
        interface enp8s0
        state MASTER
        unicast_src_ip 192.168.71.245
        unicast_peer {
                192.168.71.244
                192.168.71.246
        }

        virtual_router_id 68
        priority 200
        virtual_ipaddress {
                192.168.71.254/22
        }
}
```

```
vrrp_instance administratorzy {
        interface enp9s0
        state MASTER
        unicast_src_ip 192.168.75.245
        unicast_peer {
                192.168.75.244
                192.168.75.246
        }
        virtual_router_id 72
        priority 200
        virtual_ipaddress {
                192.168.75.254/22
        }
}
```

Stworzone zostały trzy instancję protokołu VRRP, po jednej dla każdej z podsieci. Konfiguracja dla każdego node jest bardzo zbliżona. Różnicą między nimi jest inna wartość parametru *state* oraz *priority*. Kolejno BACKUP oraz priorytet 100 i 50. Domyślnie protokół VRRP działa w trybie multicast, natomiast wykorzystanie maszyn wirtualnych wymusiło zmianę trybu pracy na unicast. Jest to spowodowane ograniczeniem możliwości przesyłania pakietów multicastowych poprzez interfejsy VTEP w KVM.

5.2 Peering z zewnętrznym routerem BGP

Rozdział 6

Porównanie rozwiązania zwirtualizowanego do fizycznego

w oparciu o przyjęte kryteria. Obecnie najpopularniejszą formą zapewnienia routingu w sieci jest wykorzystanie klasycznych routerów oraz przełączników sprzętowych. W zależności od wielkości firmy, stosuje się bardzo proste routery domowe, bądź bardziej zaawansowane produkty firm takich jak Cisco bądź Juniper. W przypadku gdy korporacja wymaga zaawansowanego routingu nie tylko dla potrzeb sieci wewnętrznej w jednej placówce, konieczne jest zakup dodatkowych urządzeń. Potrzebne są wtedy wydaje routery BGP bądź Firewalle potrafiące zestawić wydajne tunele VPN.

Obecnie coraz więcej usług jest realizowanych przez oprogramowanie zainstalowane na serwerach. Fizyczne urządzenia dedykowane pod dany rodzaj usług stają się coraz mniej popularne. W niniejszym rozdziale zostanie poddana analiza zalet oraz wad rozwiązania fizycznego do zwirtualizowanego. Przez pojęcie zwirtualizowanego rozumie się wykorzystanie zyskujących na popularności kontenerów, wraz z Orkiestratorem Kubernetes, oraz siecią realizowaną przez Calico. Kryteria porównawcze które przyjęto to:

- uniwersalność
- koszt wdrożenia
- koszt utrzymania
- łatwość konfiguracji
- możliwość rozbudowy

6.1 Uniwersalność

Routery sprzętowe popularnych producentów obsługują ogromną liczbę protokołów które przez swoje specyficzne wymagania mogą być niedostępne dla rozwiązania zwirtualizowanego. Jednym z takich protokołów jest MPLS, zatem w przypadku konieczności obsługi wspomnianego protokołu zalecany jest wybór rozwiązania fizycznego.

W przypadku analizy uniwersalności, jako możliwość zastosowania sprzętu w wielu dziedzinach zdecydowaną przewagę uzyskują rozwiązania zwirtualizowane. Zakupując router sprzętowy jego funkcjonalność ogranicza się jedynie do sprawowania tej jednej funkcji. Wszelka integracja z systemami wewnętrznymi firmy jest bardzo skomplikowana i czasem niemożliwa. Korzystając z routera którego konfiguracja odbywa się poprzez pliki yaml, tak jak w przypadku opisywanego rozwiązania, istnieje możliwość łatwej integracji z oprogramowaniem firmy. Takim przykładem są wszelkie systemy CRM bądź zarządzania pracownikami.

Główną przewagą w aspekcie uniwersalności jest możliwość wykorzystania praktycznie dowolnego komputera klasy PC, bądź serwera. W przypadku potrzeby stworzenia sieci wewnątrz firmy możliwe jest zastosowanie nieużywanego już sprzętu bądź zakup nowego. W tym przypadku zakupione urządzenie może również służyć w innych obszarach działalności firmy.

6.2 Koszty

Analiza kosztu zakupu w oparciu o serwis allegro. Ceny podane orientacyjnie, bez konkretnych ofert. Koszty utrzymania.

6.3 Łatwość konfiguracji

bardzo subiektywne. Pokazanie różnic. Skupienie się na popularności obsługi np. CLI z IOS Cisco oraz małą popularność systemu Kubernetes.

6.4 Możliwość rozbudowy

Duże pole do popisu dla rozwiązań zwirtualizowanych.

Rozdział 7

Procedury testowe

Jakiś wstęp...

7.1 Testy wydajności

Z powodu ograniczeń technicznych i braku możliwości porównania z podobną platformą sprzętową, testy wydajności zostaną przeprowadzone tylko w środowisku zwirtualizowanym. Podczas testów stworzone będą cztery scenariusze możliwe w czasie pracy firmy. //Ewentualnie można porównać niektóre testy z komputerem fizycznym wpiętym w ten sam segment sieci.

1. Wysyłanie pliku na serwer.
2. Korzystanie z internetu.
3. Przesyłanie danych między siecią Administratorów a pracowników.
4. Czas dostępu do publicznie dostępnej strony www.

Środowisko testowe, tj. klaster Kubernetes został stworzony jako zespół maszyn wirtualnych. Również komputery pracowników to maszyny wirtualne uruchomione na tym samym komputerze. Jako maszyna supervisor posłużył zmodyfikowany komputer MacPro 4,1 Early 2009. Specyfikacja kluczowych parametrów znajduje się poniżej:

- Procesor: 2x Xeon E5520 4 Core 2.26GHz
- Pamięć ram: 4x8GB DDR3 1066 MT/s

	CPU [core]	RAM [GB]	load
supervisor	16	32	3.07
k8s-01-master	4	4	
k8s-02-master	4	4	
k8s-03-master	4	4	
k8s-04-worker	2	2	
k8s-06-worker	2	2	
k8s-05-worker	2	2	
k8s-loadbalancer	2	1	
administratorzy	1	512	
pracownicy	1	512	

- Karta sieciowa: Intel Corporation 82574L Gigabit Network Connection
- Dysk twardy: Samsung HD103SJ
- System Operacyjny: Debian 10

Maszyny wirtualne zostały stworzone przy pomocy rozwiązania KVM oraz QEMU w ich najnowszych wersjach, dostępnych w repozytorium Debian Buster. Poniżej została przedstawiona specyfikacja każdej z VM. W prawej części znajdują się również informacje na temat średniego obciążenia w spoczynku. Przez to pojęcie rozumie się normalną prace klastra wraz z uruchomionymi wszystkimi usługami. Nie są one jednak w żaden sposób wykorzystywane. Procesor w każdej z maszyn wirtualnych emuluje CPU supervisora. Kolumna load oznacza wartość parametru systemu linux *Load Averages* dla ostatnich 15 minut. Na każdej maszynie wirtualnej został zainstalowany system operacyjny Ubuntu 18.04.3 LTS bez środowiska graficznego.

7.1.1 Wysyłanie pliku na serwer

Pierwszy test symuluje wysyłanie dużego pliku na zewnętrzny serwer w internecie. Do tego celu wykorzystano narzędzie iperf3. Pomiar został wykonany do serwera umieszczonego w obrębie sieci lokalnej, do którego minimalna przepustowość łącza wynosi 1Gbps. Klienta uruchomiono na maszynie wirtualnej *pracownicy*. Parametry testu są następujące:

- Czas trwania: 30 sekund.

- Serwer docelowy: speedtest.k.pl
- Protokół: TCP
- MTU: 1500

Trasa do serwera jest następująca:

1. 192.168.71.245
2. 212.127.89.94
3. 212.127.92.118
4. 79.110.205.97

Dla porównania taki sam test przeprowadzono z komputera bezpośrednio wpiętego do sieci lokalnej, bez wykorzystania technologii wirtualizacji oraz klastra. Wyniki testów prezentują się następująco:

7.2 Testy niezawodności

Rozdział 8

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