Kubernetes Under the hood

Part II

Bartosz Kondek

Networking




Networking w podach

Kubernetes nie zarządza sieciami - daje możliwość implementacji własnego rozwiązania

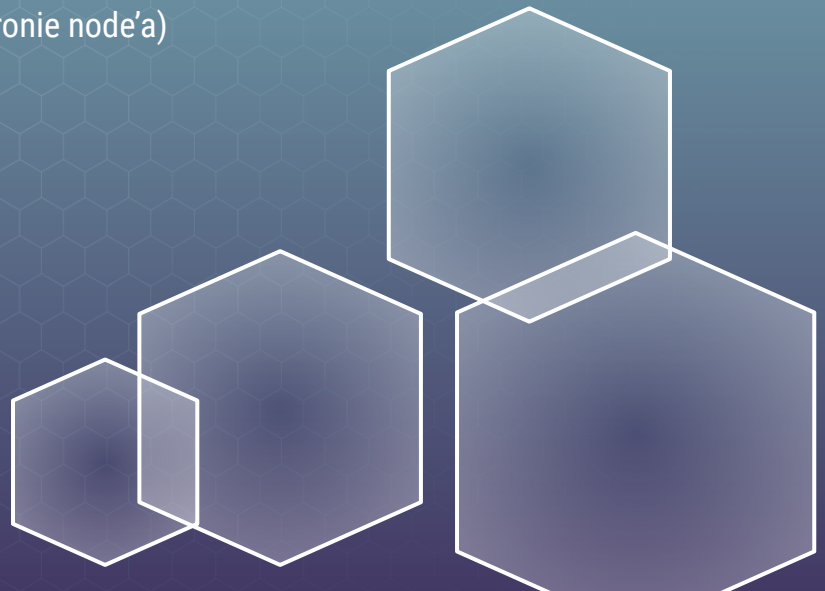
Wymagania kubernetesa odnośnie sieci:

- każdy pod musi posiadać unikalny adres IP
- Każdy pod musi mieć możliwość komunikacji z innymi podami na tym samym nodzie
- Każdy pod musi mieć możliwość komunikacji z każdym podem na innym nodzie bez NATu

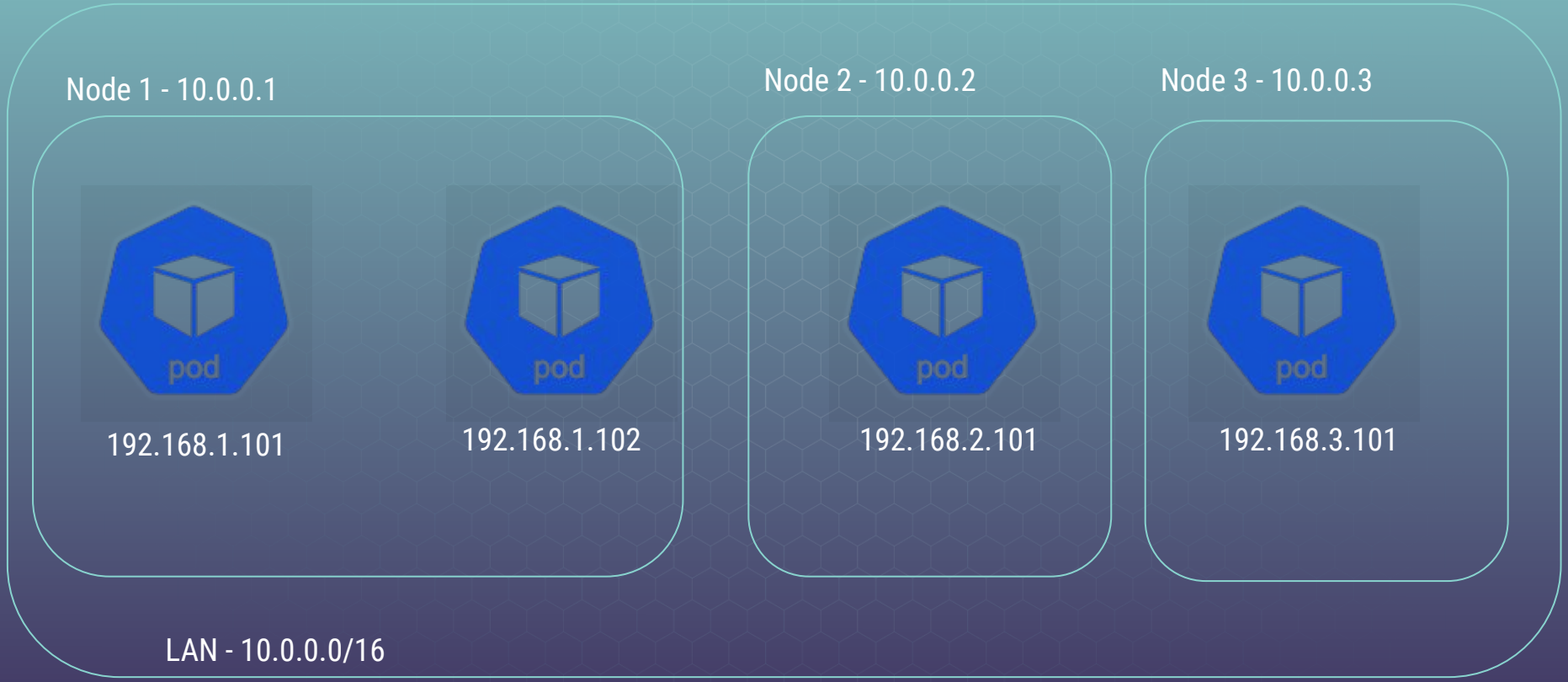
Networking w podach

Konfiguracja sieci - komunikacji poda z innymi podami w klastrze

- Utworzenie “network namespace” (izolowana sieć) i interfejsu sieciowego na hoście
- Połączenie interfejsu (hosta) z network namespace - utworzenie bridge’a
- Połączenie interfejsu poda do bridge’a
- Przypisanie unikalnego adresu IP do interfejsu poda (i po stronie node’a)
- Podniesienie interfejsu
- NAT / maskarada (+gateway)



POD networking – manualna konfiguracja



POD networking – manualna konfiguracja + GW

Node 1 - 10.0.0.1

192.168.1.101

192.168.1.102

GW do 192.168.1.0/24: 10.0.0.1

Node 2 - 10.0.0.2

192.168.2.101

GW: 10.0.0.2

Node 3 - 10.0.0.3

192.168.3.101

GW: 10.0.0.3

LAN - 10.0.0.0/16

Można to sobie oskryptować samodzielnie

CNI – Container Network Interface

CNI to standard/specyfikacja + biblioteki to pisania pluginów obsługujących konfigurację sieci

“Zadania” CNI:

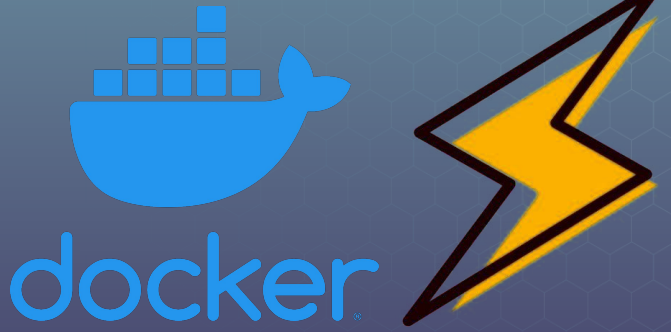
- Tworzenie network namespace podczas startu poda
- Identyfikacja sieci, do której kontener ma się podpiąć
- Wywołanie polecenia ADD podczas tworzenia poda
- Wywołanie polecenia DEL podczas usuwania poda

JSON jako format konfiguracji sieci

CNI vs CNM

CNM - Container Network Model - standard obsługi sieci wspierany przez dockera

- Nie można uruchomić “natywnie” kontenera dockera z siecią skonfigurowaną przez CNI
- Workaround: *docker run --network=none*
- Dodanie bridge’a do kontenera (CNI)



CNI – Container Network Interface – Pluginy

“Zadania” pluginów CNI:

- Musi wspierać komendy ADD/DEL/CHECK
- Musi wspierać parametry jak container id, network namespace
- Musi zarządzać przydzielaniem adresów IP do podów
- Musi zwracać wyniki w określonym formacie

Pluginy:

- Bridge
- VLAN
- IPVLAN
- MACVLAN
- Windows
- DHCP
- Host-local
- 3rd party

Konfiguracja pluginu CNI

- Konfigurowalny w kubelet na każdym nodzie
- Dostępne pluginy:
  - */opt/cni/bin*
- Aktualnie używany plugin i jego konfiguracja:
  - */etc/cni/net.d/*

```
root@kind-control-plane:/opt/cni/bin# cat /etc/cni/net.d/10-kindnet.conflist
{
        "cniVersion": "0.3.1",
        "name": "kindnet",
        "plugins": [
        {
                "type": "ptp",
                "ipMasq": false,
                "ipam": {
                        "type": "host-local",
                        "dataDir": "/run/cni-ipam-state",
                        "routes": [
                                {
                                        "dst": "0.0.0.0/0"
                                }
                        ],
                        "ranges": [
                        [
                                {
                                        "subnet": "10.244.0.0/24"
                                }
                        ]
                ]
                }
        },
        {
                "type": "portmap",
                "capabilities": {
                        "portMappings": true
                }
        }
        ]
}
```

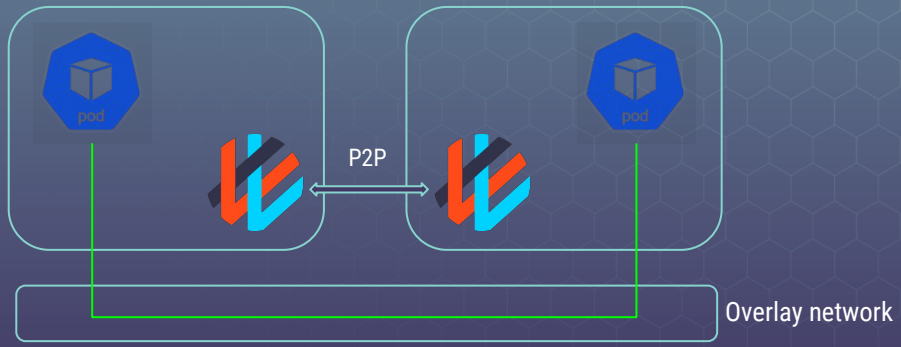
Weave Net

Flannel

Calico

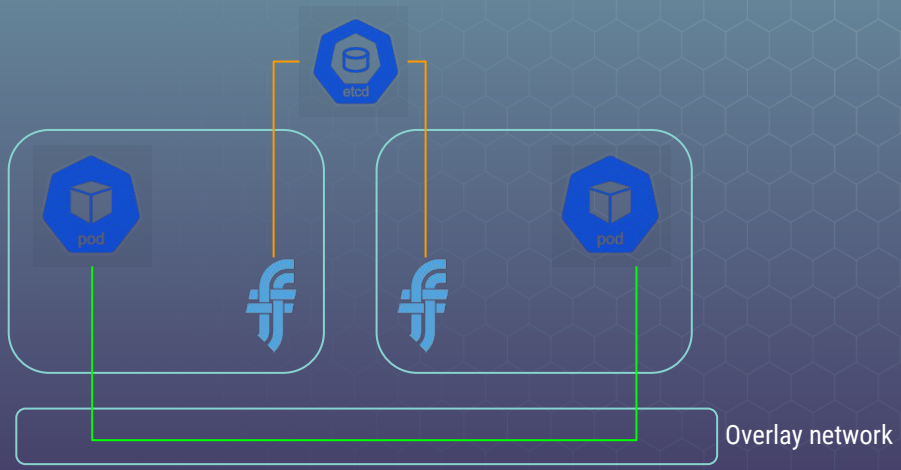
Weave Net CNI

- Pody wysyłają pakiety to adresu docelowego, nie muszą znać tras (trasa do agenta)
- Agent na każdym nodzie - komunikują się ze sobą, mają informację na którym nodzie są pody z danym adresem IP
- Enkapsulacja pakietów: adres poda -> adres node'a
- Domyślna adresacja (10.32.0.0/12) - dzielona na node'y



Flannel CNI

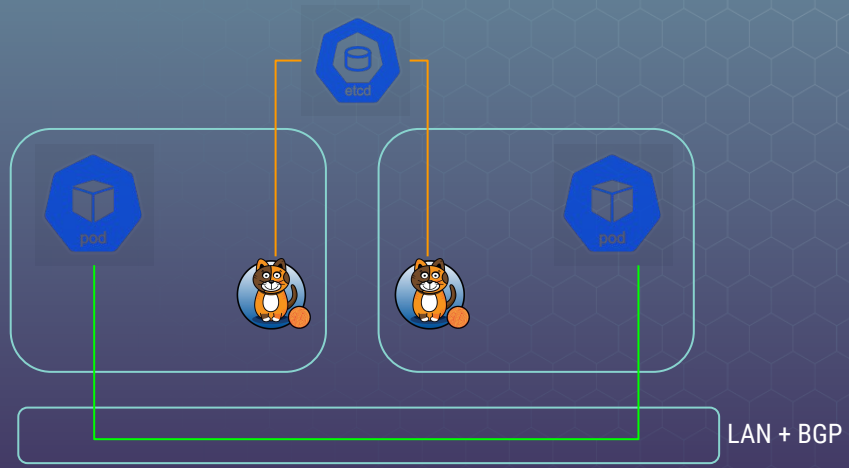
- Agent na każdym hoście alokuje adresację IP dla hosta z większej puli
- Używa Kubernetes API lub etcd do przechowywania konfiguracji sieci, podsieci
- “overlay network” (enkapsulacja)
- Nastawiony jedynie na networking (host -> host)
- brak obsługi np. Network policy

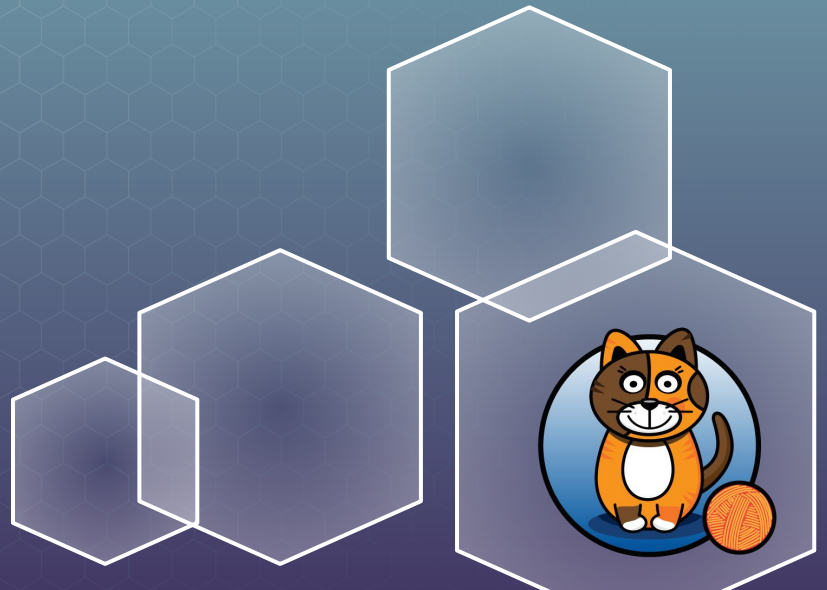


Calico CNI

- BGP zamiast “overlay network” do komunikacji między hostami
- Możliwość użycia routowalnych adresów IP dla podów
- Datastore: Kubernetes API lub etcd
- Wspiera szyfrowanie poprzez Wireguard (pod -> pod)
- Pełne wsparcie Network Policy - używa iptables
- Calico Enterprise: hierarchiczne network policy, IDS, multi-cluster management
- Wspierany przez Amazon EKS, Azure AKS, Google GKE, IBM IKS

- Domyślnie jedna podsieć dla całego klastra, ale można ją elastycznie dzielić (node selector, annotacje poda/namespace)
- Wsparcie dla dual stack lub tylko IPv6
- Dostępny również na VMki, bare metal





Azure CNI

- Podział adresacji IP na node'y planowany z góry
- Każdy node ma określoną maksymalną ilość obsługiwanych podów - na tej podstawie rezerwowana jest odpowiednia ilość adresów IP
- Pozwala to powiększać klaster bez konieczności reorganizacji adresacji IP
- Możliwa komunikacja pod <-> VM w ramach tej samej wirtualnej sieci

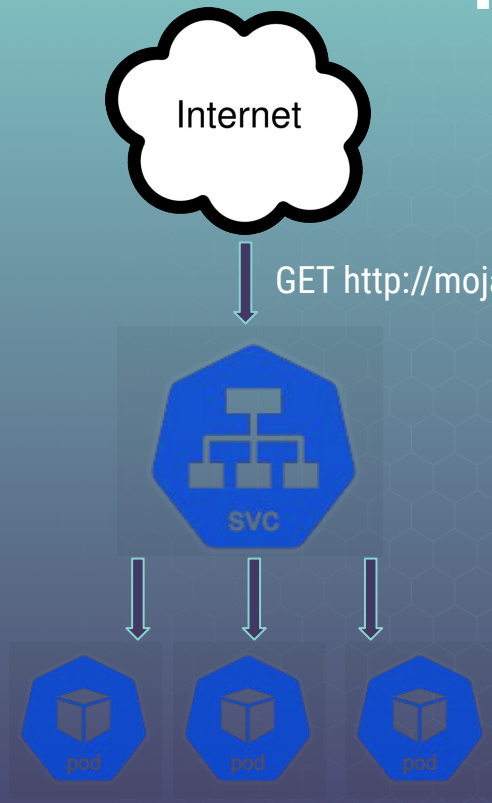
Service networking

- Serwis = wirtualny zasób = regułka w iptables (domyślnie) do przekierowanie ruchu do konkretnych podów
- Service IP - dostępne z każdego node'a/poda w klastrze
- Kubelet - zarządza tworzeniem podów; kube-proxy - sprawdza zmiany z klastrze i "aktualizuje" serwisy
- Zakres adresów IP zdefiniowany w API serwerze, nie powinien się "nakładać" z adresacją dla podów

Cluster DNS (Core DNS)

- Jest na każdym nodzie w domyślnym namespace
- Do **serwisów** można się odwoływać bezpośrednio po nazwie
- FQDN: serwis.namespace.svc.cluster.local, gdzie cluster.local do domyślna wartość dla nazwy klastra
- Do konkretnych podów można się odwołać po adresie IP (myślniki zamiast kropek), np. 10-244-2-5.namespace.pod.cluster.local - działa tylko, jeśli ustawiona jest dyrektywa “pods insecure” w konfiguracji coredns
- Każdy pod używa core-dns jako nameserver
- Każdy pod zawiera domeny wyszukiwania:
  - default.svc.cluster.local
  - svc.cluster.local
  - cluster.local

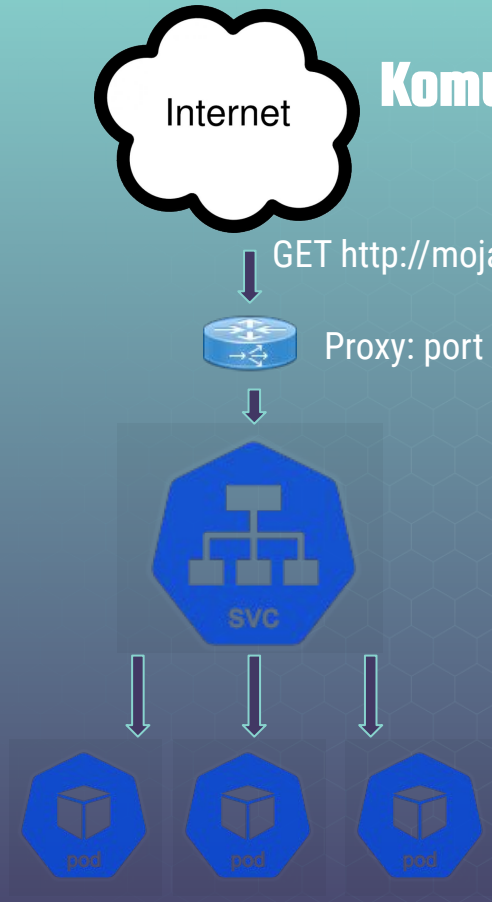
Komunikacja z internetem



Typ: NodePort

port >30000

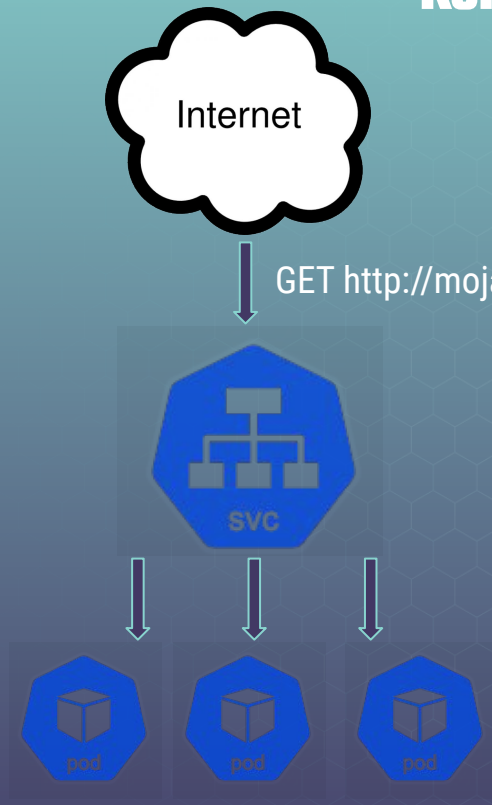
Komunikacja z internetem + proxy



Typ: ClusterIP

port >30000

Komunikacja z internetem + LB

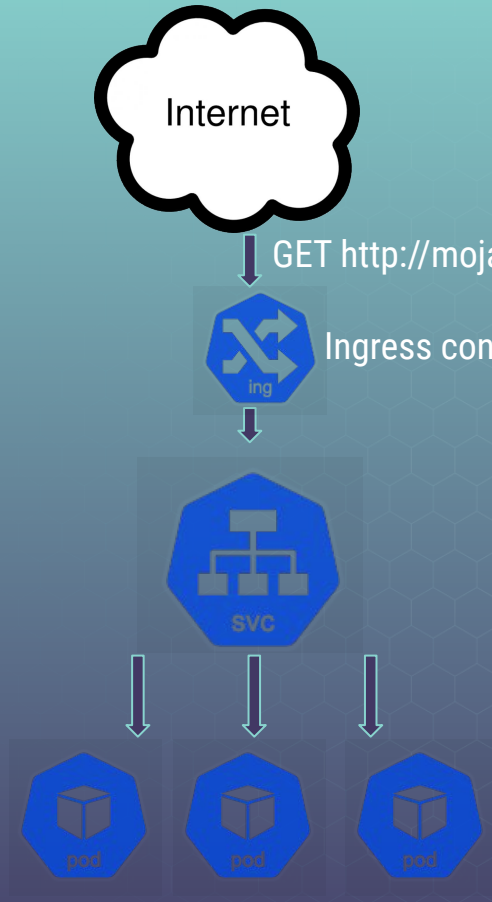


Typ: LoadBalancer

Request o adres IP dla serwisu

IP per serwis = $$$

Ingress Controller



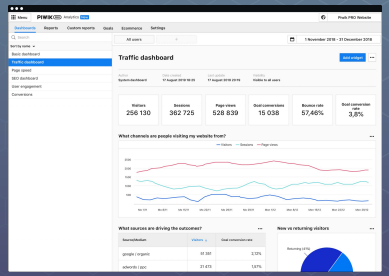
Typ: NodePort

Ingress Controller

- Ingress controller = Load Balancer w warstwie 7
- Typ: nodePort lub LoadBalancer
- Implementacja jako haproxy, nginx, itp.
- Używa “ingress resources” jako reguł kierowania ruchu do serwisów
    - Np. na podstawie domeny, ścieżki
- Konfigurowalny poprzez ConfigMap
- Default backend (default-http-backend) powinien być zdeployowany dla requestów, które nie pasują do żadnych regułek, np. do serwowania błędu 404
- Rewrite-target - annotacje:
    - haproxy.org/path-rewrite
    - nginx.ingress.kubernetes.io/rewrite-target

Ingress Controller

Internet



aplikacja

The end

Bibliografia

- Udemy - Certified Kubernetes Administrator
- https://kubernetes.io/docs/concepts/cluster-administration/networking/
- https://docs.projectcalico.org/reference/
- https://github.com/flannel-io/flannel
- https://docs.microsoft.com/en-us/azure/aks/concepts-network
- https://rancher.com/blog/2019/2019-03-21-comparing-kubernetes-cni-providers-flannel-calico-canal-and-weave/

Bartosz Kondek, 05.2021