# Sieć klastra i polityki sieciowe

**Poziom: zaawansowany** **Dotyczy: Kubernetes 1.30+** **Czas: ~25 min**

*Model sieci Kubernetesa zakłada, że każdy pod widzi każdy inny pod bez żadnych przeszkód — ta strona pokazuje, kto realnie to zapewnia i jak świadomie to ograniczamy przez NetworkPolicy.*

**Skróty użyte na tej stronie (7)**

| Skrót | Co znaczy |
|---|---|
| **API** | Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą |
| **CIDR** | Classless Inter-Domain Routing: zapis podsieci z długością maski, na przykład 10.20.10.0/24 |
| **CNI** | Container Network Interface: wtyczka, która daje sieć podom w Kubernetesie |
| **DNS** | Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP |
| **IP** | Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection) |
| **NAT** | Network Address Translation: podmiana adresów IP na granicy sieci |
| **TCP** | Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia |

# Fiszka

**TYLKO ODCZYT**

## Sprawdzenie zainstalowanej wtyczki CNI: kubectl get pods

```
kubectl get pods -n kube-system -l k8s-app=cilium
```

Sprawdzamy, czy pody wtyczki sieciowej (tu: Cilium) działają na wszystkich węzłach — nazwa etykiety i przestrzeń nazw zależą od wybranej wtyczki, opisanej niżej.

Przykład:

```
NAME              READY   STATUS    RESTARTS   AGE
cilium-2k9wp      1/1     Running   0          40d
cilium-7xqm4      1/1     Running   0          40d
```

**CNI** = Container Network Interface: wtyczka, która daje sieć podom w Kubernetesie · **kubectl** = Kubernetes control: główne narzędzie Kubernetesa

**PLIK KONFIGURACYJNY**

## Utworzenie polityki „domyślnie odrzucaj" dla przestrzeni nazw

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: domyslnie-odrzucaj
  namespace: NazwaPrzestrzeni
spec:
  podSelector: {}
  policyTypes:
    - Ingress
    - Egress
```

Blokujemy cały ruch wejściowy i wyjściowy dla wszystkich podów w przestrzeni nazw — punkt startowy, do którego dopisujemy jawne wyjątki, opisany niżej.

**PLIK KONFIGURACYJNY**

## Zezwolenie na ruch tylko z określonych podów

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: NazwaUslugi-wejscie
  namespace: NazwaPrzestrzeni
spec:
  podSelector:
    matchLabels:
      app: NazwaAplikacji
  policyTypes:
```

```
    - Ingress
  ingress:
    - from:
        - podSelector:
            matchLabels:
              app: sklep-front
      ports:
        - protocol: TCP
          port: NumerPortu
```

Otwieramy ruch przychodzący do podów z etykietą `app: NazwaAplikacji` wyłącznie od podów z etykietą `app: sklep-front`, na jednym konkretnym porcie.

**TCP** = Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia

**PLIK KONFIGURACYJNY**

## Zezwolenie na ruch wyjściowy do konkretnej przestrzeni nazw

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: NazwaUslugi-wyjscie-dns
  namespace: NazwaPrzestrzeni
spec:
  podSelector:
    matchLabels:
      app: NazwaAplikacji
  policyTypes:
    - Egress
  egress:
    - to:
        - namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: kube-system
      ports:
        - protocol: UDP
          port: 53
        - protocol: TCP
```

```
      port: 53
```

Otwieramy ruch wyjściowy do przestrzeni nazw `kube-system` wyłącznie na portach DNS — bez tego wyjątku polityka „domyślnie odrzucaj” z pierwszej fiszki blokuje też rozwiązywanie nazw, opisane na stronie DNS w klastrze i wykrywanie usług.

**DNS** = Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP · **UDP** = User Datagram Protocol: szybki protokół sieciowy bez potwierdzeń · **TCP** = Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia

**PLIK KONFIGURACYJNY**

## Zezwolenie na ruch wyjściowy do zewnętrznego bloku IP

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: NazwaUslugi-wyjscie-zewnetrzne
  namespace: NazwaPrzestrzeni
spec:
  podSelector:
    matchLabels:
      app: NazwaAplikacji
  policyTypes:
    - Egress
  egress:
    - to:
        - ipBlock:
            cidr: 203.0.113.0/24
      ports:
        - protocol: TCP
          port: 443
```

Otwieramy ruch wyjściowy do konkretnej publicznej podsieci — typowy wyjątek dla poda łączącego się z zewnętrznym API dostawcy.

**IP** = Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection) · **API** = Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą · **TCP** = Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia

**TYLKO ODCZYT**

### Diagnostyka łączności między podami: kubectl exec i curl

```
kubectl exec NazwaPoda -n NazwaPrzestrzeni -- curl -sv --max-time 5
http://NazwaUslugi.NazwaPrzestrzeni.svc.cluster.local:NumerPortu
```

Sprawdzamy z wnętrza jednego poda, czy dociera do usługi innego poda — najszybszy sposób odróżnienia awarii sieci od awarii samej aplikacji.

**kubectl** = Kubernetes control: główne narzędzie Kubernetesa

# Z życia wzięte

**Z ŻYCIA WZIĘTE**

O 8:30 aplikacja katalogu produktów przestaje odpowiadać na zapytania DNS zaraz po tym, jak zespół bezpieczeństwa wdraża nową politykę ograniczającą ruch wejściowy do tej usługi. Tomek sprawdza `kubectl get networkpolicy` i widzi, że polityka wymienia w `policyTypes` zarówno `Ingress`, jak i `Egress`, mimo że w zamyśle miała ograniczyć tylko ruch przychodzący, a brak jakiejkolwiek reguły `egress` blokuje przy okazji cały ruch wyjściowy, łącznie z zapytaniami do CoreDNS. Po usunięciu `Egress` z `policyTypes` i dodaniu osobnej, jawnej polityki dla `Ingress` aplikacja znowu rozwiązuje nazwy i łączy się z bazą bez przeszkód.

# Model sieci Kubernetesa

**Pojęcie: model sieci Kubernetesa.** Zestaw założeń, na których stoi cała reszta mechanizmów sieciowych klastra: każdy pod dostaje własny, unikalny w całym klastrze adres IP, i każdy pod może nawiązać połączenie z każdym innym podem po tym adresie bezpośrednio, bez translacji adresów (NAT) po drodze. Pod widzi adres drugiego poda dokładnie taki, jaki ten drugi pod sam o sobie zgłasza — inaczej niż w typowej sieci Dockera na pojedynczym hoście, gdzie kontenery na różnych hostach w ogóle się nie widzą bez ręcznie skonfigurowanego routingu albo publikacji portów.

To założenie upraszcza drastycznie wszystko, co dzieje się wyżej: Service, opisany na stronie Service: jak ruch trafia do poda, mógłby w ogóle nie istnieć jako osobna warstwa pośrednia, gdyby

nie to, że adresy podów są nietrwałe — sama łączność między nimi jest już zapewniona przez ten model, zanim jeszcze dojdzie do jakiegokolwiek Service.

# Czym jest CNI

> **Pojęcie: CNI (Container Network Interface).** Specyfikacja i zestaw wtyczek odpowiedzialnych za faktyczne wprowadzenie w życie modelu sieci opisanego wyżej — to CNI nadaje podom adresy IP, tworzy interfejsy sieciowe wewnątrz nich i ustawia trasy routingu, tak żeby założenie „każdy pod widzi każdy" było prawdą na poziomie prawdziwych pakietów, a nie tylko dokumentacji. Kubernetes sam z siebie nie implementuje żadnej sieci — kubelet na każdym węźle wywołuje przy starcie poda skonfigurowaną wtyczkę CNI i to ona wykonuje całą pracę.

Bez zainstalowanej wtyczki CNI pody utkną trwale w stanie `Pending` albo `ContainerCreating` — kubelet nie ma jak nadać im adresu IP, więc kontener nigdy nie startuje, mimo że reszta klastra (API, scheduler) działa poprawnie.

# Porównanie popularnych wtyczek CNI

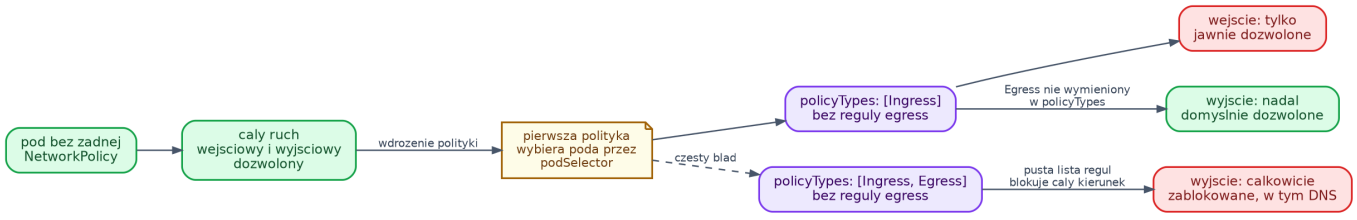
| Wtyczka | Model | Charakterystyka |
|---|---|---|
| Flannel | prosty overlay (VXLAN) | najmniej skomplikowana konfiguracja, brak natywnego wsparcia dla `NetworkPolicy` bez dodatkowego komponentu |
| Calico | routing (BGP) albo overlay (VXLAN/IP-in-IP) | dojrzałe, pełne wsparcie `NetworkPolicy`, popularna w środowiskach on-premise z własnym routingiem |
| Cilium | eBPF, bez klasycznego `iptables`/`kube-proxy` | najwyższa wydajność przy dużej skali, rozszerzone polityki sieciowe (`CiliumNetworkPolicy`) wykraczające poza standard, wbudowana obserwowalność ruchu (Hubble) |
| Weave Net | overlay z szyfrowaniem między węzłami | prostota konfiguracji zbliżona do Flannela, szyfrowanie ruchu między węzłami bez dodatkowej warstwy |

**Uważaj:** wtyczka CNI decyduje też o tym, czy `NetworkPolicy` w ogóle zadziała — sam obiekt `NetworkPolicy` jest częścią standardowego API Kubernetesa i przyjmuje się bez błędu niezależnie od zainstalowanej wtyczki, ale reguły w nim opisane egzekwuje dopiero wtyczka, która to potrafi. Klaster z samym Flannelem bez dodatkowego komponentu (np. Calico obok niego wyłącznie do polityk) przyjmie obiekt `NetworkPolicy` i będzie go całkowicie ignorował, bez żadnego ostrzeżenia.

# NetworkPolicy

> **Pojęcie: NetworkPolicy.** Obiekt Kubernetesa opisujący, jaki ruch sieciowy wejściowy i wyjściowy jest dozwolony dla wybranych podów — jedyny wbudowany mechanizm ograniczania łączności między podami w standardowym API, w kontrze do domyślnej otwartości opisanej wyżej.

**Uważaj:** dopóki żadna polityka nie wybiera danego poda, cały ruch do niego i od niego jest dozwolony — to bezpośrednia konsekwencja modelu sieci z góry tej strony i kluczowa zasada, od której zaczyna się zrozumienie całego mechanizmu. W momencie, gdy pierwsza polityka zaczyna wybierać danego poda przez `podSelector` (nawet jeśli reguła w niej opisuje coś zupełnie innego, np. tylko ruch wejściowy), Kubernetes zaczyna domyślnie blokować cały pozostały ruch tego typu, którego dana polityka dotyczy (`Ingress` albo `Egress`), i przepuszcza tylko to, co jawnie opisano. To jest najczęstsza pułapka tego mechanizmu: dodanie jednej wąskiej polityki `Ingress` do poda, który wcześniej nie miał żadnej, potrafi przy okazji odciąć mu też cały ruch wyjściowy, jeśli ta sama polityka wymienia `Egress` w `policyTypes` bez żadnej reguły `egress` — pusta lista reguł przy obecnym typie polityki oznacza całkowitą blokadę tego kierunku.




*Dopóki żadna polityka nie wybiera poda, ruch jest domyślnie otwarty w obu kierunkach, a wymienienie Egress w policyTypes bez żadnej reguły egress blokuje cały ruch wyjściowy, w tym DNS (kliknięcie otwiera pełny rozmiar)*

# Selektory: pody, przestrzenie nazw, bloki IP

Reguła `ingress` albo `egress` opisuje źródło (albo cel) ruchu przez jeden z trzech selektorów, które można też łączyć: `podSelector` (pody po etykietach, domyślnie w tej samej przestrzeni nazw), `namespaceSelector` (wszystkie pody w przestrzeniach nazw pasujących do etykiety) i `ipBlock` (zakres adresów IP w notacji CIDR, jedyny sposób na dopuszczenie ruchu spoza klastra). Połączenie `podSelector` i `namespaceSelector` w jednym wpisie `from` zawęża dopasowanie do konkretnych podów w konkretnych przestrzeniach nazw, a nie do sumy obu warunków osobno — to rozróżnienie (jeden wpis kontra dwa osobne wpisy w liście `from`) jest źródłem błędów, które przepuszczają więcej ruchu niż zamierzono.

# Diagnostyka łączności między podami

Kolejność sprawdzania, gdy dwa pody, które powinny się widzieć, nie widzą się:

1. Sprawdzamy, czy oba pody w ogóle są w stanie `Running` i `Ready` (`kubectl get pods -o wide`).
2. Sprawdzamy łączność z wnętrza jednego poda do adresu IP drugiego bezpośrednio, z pominięciem DNS i Service, żeby wykluczyć te dwie warstwy z podejrzeń.
3. Sprawdzamy, czy istnieje `NetworkPolicy` obejmująca któregoś z podów (`kubectl get networkpolicy -n NazwaPrzestrzeni`) i czy jej reguły faktycznie dopuszczają ten konkretny ruch.
4. Sprawdzamy stan podów wtyczki CNI na obu węzłach, na których żyją nasze pody — awaria CNI na jednym konkretnym węźle tłumaczy asymetryczne problemy z łącznością (działa do jednych podów, nie działa do innych).

```
kubectl exec NazwaPoda -n NazwaPrzestrzeni -- nc -zv -w 3 10.244.2.7 NumerPortu
```

Sprawdzamy gołe połączenie TCP do adresu IP drugiego poda, z pominięciem nazwy DNS i Service — jeśli to się nie udaje, problem leży w samej sieci albo w `NetworkPolicy`, a nie w warstwach wyższych.

# Częste błędy

**Przekonanie, że NetworkPolicy działa bez wtyczki CNI, która ją wspiera.** Obiekt zostaje przyjęty przez API bez błędu i sprawia wrażenie aktywnej ochrony, podczas gdy realnie żaden pakiet nie jest przez nią zatrzymywany — jedynym sposobem, żeby się upewnić, jest świadome sprawdzenie dokumentacji zainstalowanej wtyczki.

**Polityka Ingress blokująca przy okazji Egress.** Dodanie `Egress` do `policyTypes` bez choćby jednej reguły `egress` blokuje cały ruch wyjściowy poda, łącznie z zapytaniami DNS — objawem jest aplikacja, która przestaje cokolwiek rozwiązywać i łączyć się na zewnątrz zaraz po wdrożeniu polityki dotyczącej z pozoru tylko ruchu wejściowego.

**Brak wyjątku na DNS przy polityce „domyślnie odrzucaj”.** Klasyczny efekt uboczny pierwszej fiszki tej strony — bez osobnej reguły otwierającej port 53 do `kube-system` cała przestrzeń nazw traci możliwość rozwiązywania jakichkolwiek nazw, włącznie z nazwami usług we własnej przestrzeni.

# □□ Ćwiczenia

> Ćwiczenia zakładają klaster testowy z wtyczką CNI wspierającą `NetworkPolicy` (Calico albo Cilium) i dwa pody w tej samej przestrzeni nazw.

# Ćwiczenie 1 — efekt polityki domyślnie odrzucaj

```
kubectl exec pod-a -n NazwaPrzestrzeni -- curl -sv --max-time 3 http://pod-b-ip:80
kubectl apply -f domyslnie-odrzucaj.yaml
kubectl exec pod-a -n NazwaPrzestrzeni -- curl -sv --max-time 3 http://pod-b-ip:80
```

Sprawdzamy łączność między dwoma podami przed i po wdrożeniu polityki „domyślnie odrzucaj” dla całej przestrzeni nazw.

**Odpowiedź**

Pierwsze połączenie się udaje — bez żadnej polityki ruch jest domyślnie dozwolony. Po wdrożeniu polityki drugie połączenie kończy się przekroczeniem czasu oczekiwania, bo polityka z pustą listą reguł blokuje cały ruch tego typu dla wszystkich podów w przestrzeni nazw.

# Pytania kontrolne

**1. Dlaczego pod bez żadnej dedykowanej dla niego polityki NetworkPolicy może mimo to stracić łączność po wdrożeniu polityki gdzieś indziej w tej samej przestrzeni nazw?**

**Odpowiedź**

Jeśli ta polityka wybiera pody przez pusty `podSelector: {}`, obejmuje wszystkie pody w przestrzeni nazw, nie tylko te, dla których pisaliśmy ją w zamyśle.

**2. Czym różni się blokada ruchu przez NetworkPolicy od blokady przez firewall systemu operacyjnego na węźle?**

**Odpowiedź**

NetworkPolicy działa na poziomie tożsamości poda (etykiety, przestrzeń nazw) i jest egzekwowana przez wtyczkę CNI, niezależnie od tego, na którym węźle pod aktualnie żyje — reguła podąża za podem przy jego przeniesieniu. Firewall na węźle operuje na adresach IP i interfejsach konkretnej maszyny, bez wiedzy o tym, czym jest pod ani do którego namespace'u należy.

**Powiązane:** Service: jak ruch trafia do poda · DNS w klastrze i wykrywanie usług · Diagnostyka: objaw, przyczyna, pierwsze kroki

**Źródła:** dokumentacja kubernetes.io, rozdziały „Cluster Networking", „Network Policies" i „Network Plugins" · dokumentacja projektów Cilium, Calico i Flannel

---

Revision #33
Created 2026-09-25 10:33:05 UTC by Tomasz Konieczny
Updated 2026-10-26 14:17:07 UTC by Tomasz Konieczny