Service: jak ruch trafia do poda

Poziom: średni **Dotyczy: Kubernetes 1.30+** **Czas: ~20 min**

Service daje podom stały adres i stałą nazwę, dzięki czemu reszta klastra może się z nimi komunikować mimo że same pody znikają i pojawiają się na nowo z innym adresem IP.

Skróty użyte na tej stronie (3)

Skrót	Co znaczy
API	Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą
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)

Fiszka

TYLKO ODCZYT

Wylistowanie usług w przestrzeni nazw: kubectl get svc

```
kubectl get svc -n NazwaPrzestrzeni
```

Sprawdzamy, jakie usługi istnieją w danej przestrzeni nazw, jaki mają typ, jaki adres ClusterIP i na jakich portach nasłuchują.

Przykład:

```
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
sklep-api     ClusterIP   10.96.142.7     <none>        8080/TCP       14d
sklep-front   NodePort    10.96.201.33    <none>        80:31080/TCP   14d
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

PLIK KONFIGURACYJNY

Utworzenie usługi typu ClusterIP

```
apiVersion: v1
kind: Service
metadata:
  name: NazwaUslugi
  namespace: NazwaPrzestrzeni
spec:
  type: ClusterIP
  selector:
    app: NazwaAplikacji
  ports:
    - port: NumerPortu
      targetPort: NumerPortu
      protocol: TCP
```

```
kubectl apply -f service-clusterip.yaml
```

Tworzymy usługę dostępną tylko wewnątrz klastra, adresowaną przez pozostałe pody pod nazwą NazwaUslugi — dokładny mechanizm rozwiązywania tej nazwy opisuje strona DNS w klastrze i wykrywanie usług.

DNS = Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP ·**TCP** = Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia ·**kubectl** = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Podejrzenie, które pody stoją za usługą: kubectl get endpoints

```
kubectl get endpoints NazwaUslugi -n NazwaPrzestrzeni
```

Sprawdzamy listę adresów IP i portów podów faktycznie dobranych przez selektor usługi — to jest pierwsze miejsce, w którym widzimy, czy usługa w ogóle ma do czego kierować ruch.

Przykład:

```
NAME        ENDPOINTS                           AGE
sklep-api   10.244.1.12:8080,10.244.2.7:8080    14d
```

IP = Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection) · **kubectl** = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Podejrzenie tego samego zestawu w nowszym formacie: kubectl get endpointslices

```
kubectl get endpointslices -n NazwaPrzestrzeni -l kubernetes.io/service-name=NazwaUslugi
```

Sprawdzamy adresy podów w formacie, który realnie stoi pod maską usługi od wersji Kubernetesa 1.21 wzwyż, opisanym niżej.

Przykład:

```
NAME               ADDRESSTYPE   PORTS   ENDPOINTS                 AGE
sklep-api-x7k2p     IPv4          8080    10.244.1.12,10.244.2.7    14d
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

PLIK KONFIGURACYJNY

Utworzenie usługi typu NodePort

```
apiVersion: v1
kind: Service
metadata:
  name: NazwaUslugi
  namespace: NazwaPrzestrzeni
spec:
  type: NodePort
  selector:
    app: NazwaAplikacji
  ports:
    - port: NumerPortu
      targetPort: NumerPortu
      nodePort: 31080
```

```
    protocol: TCP
```

Otwieramy port na każdym węźle klastra, przez który ruch trafia dalej do usługi — zakres portów NodePort i sytuacje, w których po niego sięgamy, opisujemy niżej w tabeli typów.

TCP = Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia

TYLKO ODCZYT

Diagnostyczne przekierowanie portu na usługę: kubectl port-forward

```
kubectl port-forward svc/NazwaUslugi NumerPortu:NumerPortu -n NazwaPrzestrzeni
```

Otwieramy tymczasowy tunel z naszej stacji roboczej do usługi wewnątrz klastra, ważny tylko na czas działania tej komendy.

Przykład:

```
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

OSTROŻNIE · NIEODWRACALNE

Usunięcie usługi

```
kubectl delete svc NazwaUslugi -n NazwaPrzestrzeni
```

Usuwamy obiekt Service — pody, które ona adresowała, dalej działają, tracą jedynie stały punkt wejścia.

kubectl = Kubernetes control: główne narzędzie Kubernetesa

Z życia wzięte

Z ŻYCIA WZIĘTE

> O 10:40 aplikacja katalogu produktów na app01 przestaje odpowiadać na zapytania do wewnętrznej usługi cenowej, mimo że oba komponenty ewidentnie działają w `kubectl get pods`. Tomek sprawdza kod aplikacji i widzi, że ktoś kiedyś na szybko wpisał na sztywno adres IP jednego z podów usługi cenowej zamiast jej nazwy, a ten pod zniknął przy ostatniej aktualizacji obrazu i dostał nowy adres. Po zmianie adresu na nazwę usługi (`Service`) połączenie przestaje się psuć przy każdym kolejnym wdrożeniu, niezależnie od tego, które konkretnie pody akurat za nią stoją.

Po co Service istnieje

> **Pojęcie: Service.** Obiekt Kubernetesa, który daje stabilny adres IP i stabilną nazwę DNS zestawowi podów wybranych po etykiecie, niezależnie od tego, ile tych podów aktualnie żyje i pod jakimi adresami. Bez Service jedyny sposób, żeby jeden pod dogadał się z drugim, to znajomość jego adresu IP — a ten adres jest tymczasowy.

Każdy pod w Kubernetesie dostaje własny adres IP w momencie startu i traci go bezpowrotnie w momencie, w którym pod znika — restart poda po awarii węzła, aktualizacja obrazu przez Deployment albo zwykłe przeskalowanie tworzy nowy pod z nowym adresem, nie wskrzesza starego. Aplikacja, która zapamiętała adres IP innego poda i próbuje się z nim łączyć bezpośrednio, prędzej czy później zaczyna łączyć się donikąd — a najgorsze w tym jest to, że działa to poprawnie przez tygodnie, aż do pierwszego restartu.

Service rozwiązuje ten problem jedną warstwą pośrednią: zamiast pamiętać adres poda, zapamiętujemy nazwę usługi. Kubernetes sam utrzymuje aktualną listę adresów podów stojących za tą nazwą i sam kieruje na nie ruch, także wtedy, gdy pody są wymieniane w tle.

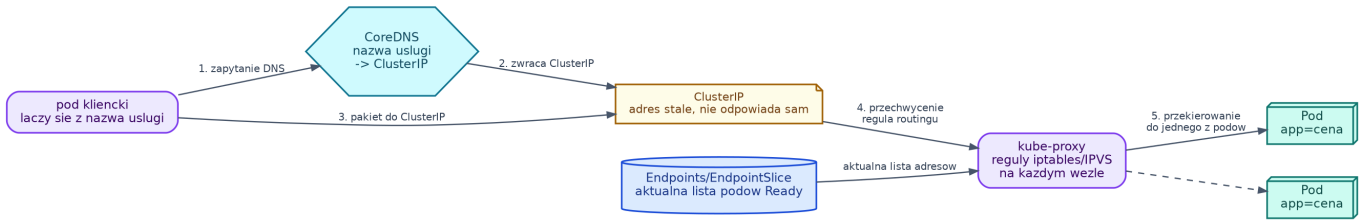
Jak to działa: selektor i Endpoints

Service łączy się z podami przez selektor etykiet — te same etykiety, które nosi Deployment albo StatefulSet zarządzający podami. Kontroler endpointów w płaszczyźnie sterowania bez przerwy obserwuje, które pody pasują do selektora danej usługi i są w stanie gotowości (`Ready`), i na bieżąco aktualizuje listę ich adresów.

> **Pojęcie: Endpoints i EndpointSlice.** Endpoints to starszy obiekt trzymający płaską listę adresów IP i portów podów pasujących do selektora usługi — jeden obiekt na całą usługę, niezależnie od liczby podów. EndpointSlice to jego następca, wprowadzony jako stabilny od wersji 1.21, który dzieli tę samą listę na mniejsze kawałki po maksymalnie 100 adresów, żeby aktualizacja jednego poda w usłudze z tysiącem replik nie wymagała przepisania

> całego, ogromnego obiektu za każdym razem. Dla nas jako administratorów oba obiekty pokazują to samo: rzeczywisty, aktualny zestaw adresów, do których usługa kieruje ruch.

Ruch do usługi nie przechodzi jednak przez żaden proces pośredniczący na ścieżce pakietu — `kube-proxy`, uruchomiony jako demon na każdym węźle, na bieżąco przekłada adres i port usługi na reguły systemu operacyjnego (`iptables` albo `IPVS`, w zależności od trybu), które przekierowują pakiet bezpośrednio do adresu jednego z podów. Adres ClusterIP usługi nigdy nie odpowiada sam — to wpis routingu na każdym węźle sprawia, że pakiet wysłany na ten adres ląduje na jednym z realnych podów.



Pod kliencki rozwiązuje nazwę usługi przez DNS na ClusterIP, a kube-proxy na podstawie aktualnej listy Endpoints przekierowuje pakiet do jednego z żywych podów (kliknięcie otwiera pełny rozmiar)

Typy Service

Typ	Jak wygląda ruch	Kiedy sięgamy
`ClusterIP` (domyślny)	stały adres widoczny tylko wewnątrz klastra	komunikacja między usługami wewnątrz klastra — zdecydowana większość przypadków
`NodePort`	dodatkowo ten sam port otwarty na każdym węźle klastra	szybki dostęp z zewnątrz bez zewnętrznego load balancera, testy, środowiska bez chmury publicznej
`LoadBalancer`	dodatkowo zewnętrzny load balancer chmury z publicznym adresem	publikacja usługi w chmurze publicznej, gdzie kontroler chmury sam tworzy load balancer
`ExternalName`	brak selektora i brak proxy — zwraca rekord `CNAME` na zewnętrzną nazwę	odwołanie się z klastra do usługi poza nim (np. zewnętrznej bazy danych) tą samą nazwą DNS co do usługi wewnętrznej

Uważaj: `LoadBalancer` bez kontrolera chmury (np. w klastrze on-premise postawionym `kubeadm`) zostaje trwale w stanie `<pending>` na polu `EXTERNAL-IP` — sam typ nic nie tworzy, potrzebuje

kontrolera, który umie rozmawiać z konkretną infrastrukturą (chmurą albo np. MetalLB w środowisku lokalnym).

Service bez selektora

Service nie musi mieć selektora wcale — wtedy Kubernetes nie tworzy dla niej automatycznie żadnych endpointów, a my utrzymujemy je ręcznie:

```
apiVersion: v1
kind: Service
metadata:
  name: NazwaUslugi
  namespace: NazwaPrzestrzeni
spec:
  ports:
    - port: NumerPortu
      protocol: TCP
---
apiVersion: v1
kind: Endpoints
metadata:
  name: NazwaUslugi
  namespace: NazwaPrzestrzeni
subsets:
  - addresses:
      - ip: 203.0.113.10
    ports:
      - port: NumerPortu
```

> **Pojęcie: Service bez selektora.** Wzorzec przydatny, gdy chcemy adresować z klastra coś, co żyje poza nim — bazę danych na fizycznym serwerze, usługę na innej platformie — tą samą nazwą DNS, jakiej używalibyśmy dla zwykłej usługi wewnętrznej. Aplikacja wewnątrz klastra łączy się z `NazwaUslugi`, nie wiedząc, że po drugiej stronie stoi adres spoza klastra.

Headless Service

> **Pojęcie: headless Service.** Usługa z `clusterIP: None` — nie dostaje własnego, stałego adresu IP, więc zapytanie DNS o jej nazwę zwraca nie jeden adres pośredniczący, tylko listę adresów IP wszystkich podów pasujących do selektora, każdego z osobna.

```
apiVersion: v1
kind: Service
metadata:
  name: NazwaUslugi
  namespace: NazwaPrzestrzeni
spec:
  clusterIP: None
  selector:
    app: NazwaAplikacji
  ports:
    - port: NumerPortu
```

Headless Service jest tym, czego realnie używa StatefulSet: każda replika w StatefulSecie ma stałą tożsamość sieciową w postaci własnej nazwy DNS (`nazwa-0.NazwaUslugi`, `nazwa-1.NazwaUslugi`), a nie wspólny, uśredniony adres. Ma to znaczenie dla baz danych i innych systemów rozproszonych, w których klient musi połączyć się z konkretnym egzemplarzem — np. wyłącznie z liderem klastra replikacji — zamiast trafić losowo na dowolną replikę przez zwykły ClusterIP.

kubectl port-forward jako narzędzie diagnostyczne

`kubectl port-forward` otwiera tunel na czas życia jednej komendy w naszym terminalu — to narzędzie do sprawdzenia, czy usługa w ogóle odpowiada, zanim zaczniemy szukać przyczyny gdzie indziej, a nie sposób na trwałe udostępnienie usługi. Tunel znika, gdy zamykamy terminal albo przerywamy komendę `Ctrl+C`, i nie przechodzi przez żaden z mechanizmów opisanych w tabeli typów wyżej — działa nawet dla usługi typu `ClusterIP`, bo cały ruch tuneluje się przez połączenie do API Kubernetesa, z pominięciem sieci klastra.

Nasz przykład: kolorowy nginx

Ten sam mechanizm selektora i Endpoints stoi za usługą `kolorowy-nginx`, opisaną na stronie k3s w laboratorium: pierwszy deploy, nginx w kolorach. Trzy pody nginksa noszą tam tę samą etykietę `app: kolorowy-nginx`

, a Service o tej nazwie kieruje na nie ruch, nie wiedząc nic ani o kolorach, ani o nazwach poszczególnych podów. Kolejne osobne połączenia trafiają losowo do innego z adresów z listy Endpoints tej usługi, co jest najprostszym możliwym sposobem, żeby zobaczyć działanie `kube-proxy` gołym okiem zamiast tylko na papierze.

```
apiVersion: v1
kind: Service
metadata:
  name: kolorowy-nginx
  namespace: kolory
spec:
  type: ClusterIP
  selector:
    app: kolorowy-nginx
  ports:
    - port: 80
      targetPort: 80
```

Częste błędy

Selektor usługi nie pasuje do żadnego poda. `kubectl get endpoints` pokazuje puste pole `<none>` mimo że pody wyglądają na działające — najczęstszą przyczyną jest literówka w etykiecie albo selektor skopiowany z innej usługi bez dostosowania.

Port usługi pomylony z portem kontenera. `port` to numer, pod którym usługa nasłuchuje, `targetPort` to numer portu, na którym faktycznie nasłuchuje kontener — jeśli aplikacja w kontenerze słucha na innym porcie niż podany w `targetPort`, usługa istnieje, endpointy istnieją, a każde połączenie kończy się odmową po stronie poda.

Oczekiwanie, że LoadBalancer zadziała bez kontrolera chmury. Pole `EXTERNAL-IP` zostaje w stanie `<pending>` bezterminowo w klastrze bez zintegrowanego dostawcy chmury albo bez lokalnego kontrolera typu MetalLB.

□□ Ćwiczenia

> Ćwiczenia zakładają dostęp do klastra testowego (np. `kind` albo `minikube`) i wdrożony przykładowy Deployment z co najmniej dwiema replikami.

Ćwiczenie 1 — obserwacja zmiany adresu poda

```
kubectl get pods -n NazwaPrzestrzeni -o wide
kubectl delete pod NazwaPoda -n NazwaPrzestrzeni
kubectl get pods -n NazwaPrzestrzeni -o wide
```

Zapisujemy adres IP jednego z podów, usuwamy go, po ponownym wystawieniu przez kontroler porównujemy nowy adres ze starym.

Odpowiedź

Nowy pod dostaje inny adres IP niż usunięty — sama usługa (jeśli jest) i jej adres ClusterIP pozostają bez zmian, bo to warstwa pośrednia, o którą chodzi na tej stronie.

Pytania kontrolne

1. Dlaczego usługa typu ClusterIP działa dla podów wewnątrz klastra, ale nie da się jej otworzyć bezpośrednio z przeglądarki poza klastrem?

Odpowiedź

Adres ClusterIP istnieje wyłącznie jako wpis w regułach routingu (`iptables` / `IPVS`) utrzymywanych przez `kube-proxy` na węzłach klastra — poza siecią klastra ten adres nikomu nic nie mówi, bo żaden węzeł spoza klastra nie ma takich reguł.

2. Po co w ogóle istnieje EndpointSlice, skoro Endpoints robi to samo?

Odpowiedź

Endpoints trzyma całą listę adresów w jednym obiekcie, więc przy usłudze z bardzo dużą liczbą replik każda drobna zmiana (jeden pod wchodzi lub wychodzi z gotowości) wymusza przesłanie całego, dużego obiektu do każdego obserwującego komponentu. EndpointSlice dzieli tę listę na kawałki, więc zmiana jednego adresu porusza tylko jeden mały kawałek.

Powiązane: DNS w klastrze i wykrywanie usług · Ingress i Gateway API: wejście z zewnątrz · Diagnostyka: objaw, przyczyna, pierwsze kroki

Źródła: dokumentacja kubernetes.io, rozdziały „Service", „EndpointSlice" i „Service Internals"

Revision #35
Created 2026-09-25 10:33:01 UTC by Tomasz Konieczny
Updated 2026-10-26 14:17:02 UTC by Tomasz Konieczny