DNS w klastrze i wykrywanie usług

◫◫ Poziom: średni **◫◫ Dotyczy: Kubernetes 1.30+, CoreDNS 1.11+** **⏱ Czas: ~15 min**

Każda usługa w klastrze dostaje własną nazwę DNS rozwiązywalną z wnętrza dowolnego poda — ta strona pokazuje, jak wygląda ten schemat nazw, co go obsługuje i jak diagnozujemy, kiedy przestaje działać.

◫◫ Skróty użyte na tej stronie (4)

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
FQDN	Fully Qualified Domain Name: pełna nazwa hosta razem z domeną, np. docker01.ad.zielona.example
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)

Fiszka

◫◫ OSTROŻNIE · NIEODWRACALNE

Sprawdzenie rozwiązywania nazwy usługi z wnętrza poda: nslookup

```
kubectl run diagnostyka --rm -it --image=busybox:1.36 --restart=Never -- nslookup
NazwaUslugi.NazwaPrzestrzeni.svc.cluster.local
```

Uruchamiamy jednorazowy pod diagnostyczny i sprawdzamy, czy nazwa usługi rozwiązuje się na adres ClusterIP — usuwa się sam po zakończeniu (`--rm`).

Przykład:

```
Server:    10.96.0.10
Address:   10.96.0.10:53

Name:      sklep-api.produkcja.svc.cluster.local
Address:   10.96.142.7
```

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

□□ TYLKO ODCZYT

Sprawdzenie pliku resolv.conf poda: cat

```
kubectl exec NazwaPoda -n NazwaPrzestrzeni -- cat /etc/resolv.conf
```

Sprawdzamy, na jaki adres serwera DNS pod jest skierowany i jaka wartość `ndots` została mu wstrzyknięta — obie sprawy opisujemy niżej.

Przykład:

```
nameserver 10.96.0.10
search produkcja.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
```

□□ **DNS** = Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP · **kubectl** = Kubernetes control: główne narzędzie Kubernetesa

□□ TYLKO ODCZYT

Podejrzenie usługi CoreDNS: kubectl get pods

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

Sprawdzamy, czy pody CoreDNS w ogóle działają — bez nich żadna nazwa w klastrze się nie rozwiąże, nawet jeśli reszta klastra wygląda zdrowo.

Przykład:

```
NAME                       READY   STATUS    RESTARTS   AGE
coredns-6f9c4b8d5f-2xk9m   1/1     Running   0          21d
coredns-6f9c4b8d5f-p7qwl   1/1     Running   0          21d
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Podgląd konfiguracji CoreDNS: kubectl get configmap

```
kubectl get configmap coredns -n kube-system -o yaml
```

Sprawdzamy plik `Corefile`, czyli faktyczną konfigurację, według której CoreDNS rozwiązuje zapytania — miejsce, do którego zaglądamy, gdy ktoś dodał niestandardowe przekierowanie dla domeny wewnętrznej firmy.

kubectl = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Sprawdzenie logów CoreDNS: kubectl logs

```
kubectl logs -n kube-system -l k8s-app=kube-dns --tail=100
```

Sprawdzamy błędy rozwiązywania nazw i ewentualne przeciążenie — powtarzające się `SERVFAIL` dla tej samej nazwy wskazuje na konkretny problem opisany niżej, a nie na przypadkową awarię sieci.

kubectl = Kubernetes control: główne narzędzie Kubernetesa

Z życia wzięte

Z ŻYCIA WZIĘTE

O 9:50 nowo wdrożona usługa raportowania na klastrze produkcyjnym nie może połączyć się z bazą kolejki w przestrzeni `cache`, mimo że obie przestrzenie są w tym samym klastrze i baza działa. Tomek sprawdza kod i widzi, że deweloper skopiował adres usługi z przykładu w innej przestrzeni nazw, więc krótka nazwa `baza-kolejki` bez dopisku przestrzeni po prostu nie istnieje tam, gdzie próbuje jej szukać resolwer poda. Po dopisaniu pełnej nazwy `baza-kolejki.cache` połączenie działa od razu, bez żadnej zmiany po stronie samej bazy.

Czym jest CoreDNS

> **Pojęcie: CoreDNS.** Serwer DNS działający jako zwykły Deployment wewnątrz klastra, w przestrzeni nazw `kube-system`, który odpowiada na zapytania DNS ze wszystkich podów w klastrze — to on tłumaczy nazwę usługi na jej adres ClusterIP i to przez niego przechodzi każde zapytanie o domenę spoza klastra wysłane z dowolnego poda. Usługa `kube-dns` (nazwa zachowana z czasów, gdy rolę tę pełnił inny serwer) wystawia CoreDNS pod stałym adresem, wpisanym automatycznie do `/etc/resolv.conf` każdego poda.

Każdy nowo utworzony obiekt Service dostaje wpis w CoreDNS automatycznie, bez naszej ingerencji — kontroler CoreDNS obserwuje API Kubernetesa i na bieżąco aktualizuje odpowiedzi na zapytania, w tym samym rytmie, w jakim usługi i pody powstają i znikają.

Schemat nazw

Pełna, w pełni kwalifikowana nazwa usługi w klastrze ma zawsze tę samą budowę:

```
NazwaUslugi.NazwaPrzestrzeni.svc.cluster.local
```

> **Pojęcie: cluster.local.** Domena bazowa klastra — domyślna wartość, którą można zmienić przy instalacji klastra, ale w zdecydowanej większości instalacji zostaje bez zmian. Segment `svc` odróżnia usługi od innych typów rekordów, jakie CoreDNS potrafi wystawić (np. `pod` dla adresów samych podów).

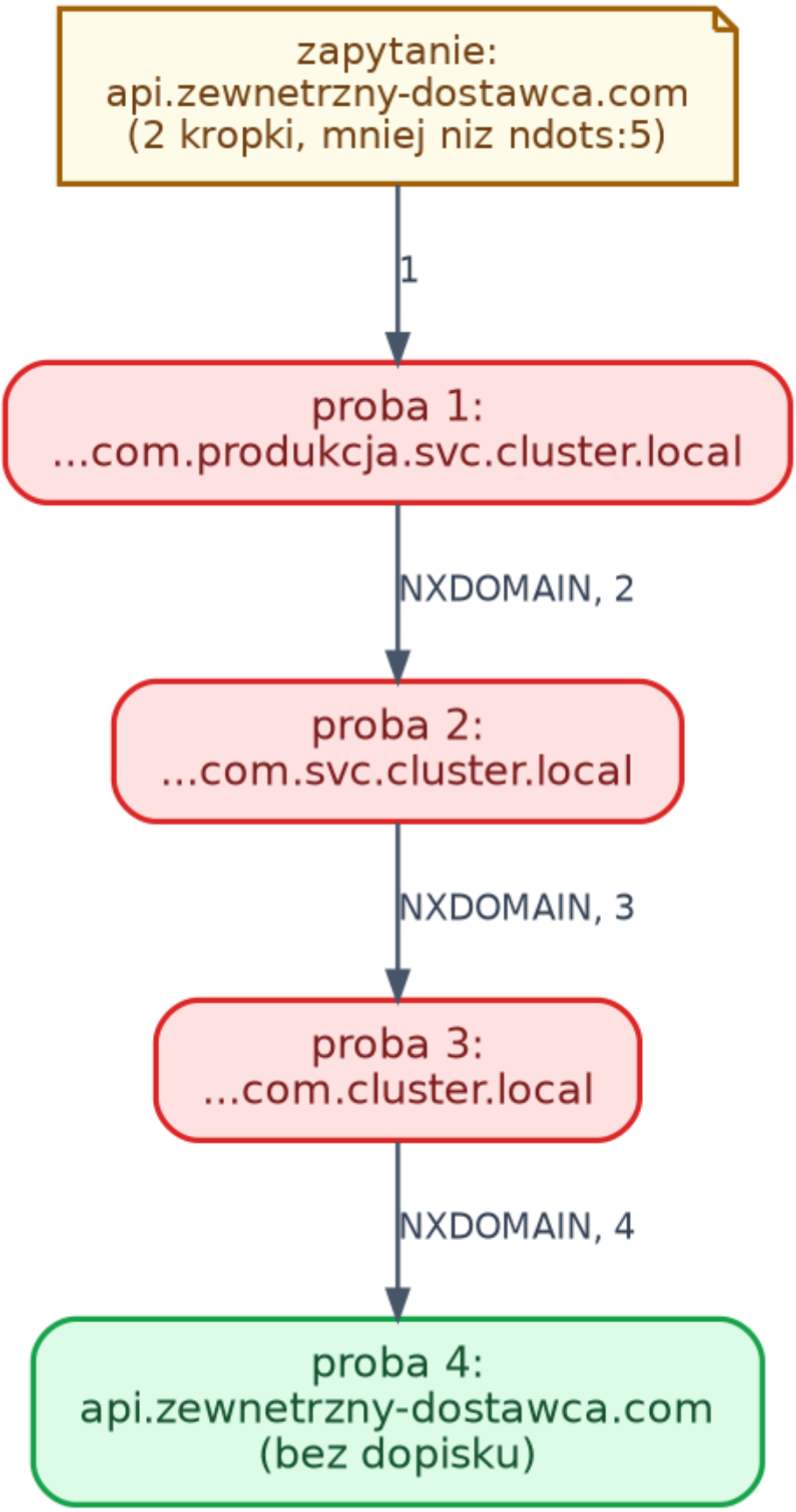
Z tego schematu wynika wprost, jak wygląda komunikacja między przestrzeniami nazw: pod w przestrzeni `produkcja` odwołujący się do usługi `sklep-api` w tej samej przestrzeni może użyć samej krótkiej nazwy `sklep-api` — mechanizm `search` z `resolv.conf`, pokazany w Fiszce, dokleja resztę. Ten sam pod odwołujący się do usługi `redis` żyjącej w przestrzeni `cache` musi użyć co najmniej `redis.cache`, bo krótka nazwa `redis` w przestrzeni `produkcja` nie istnieje. Pomijanie przestrzeni nazw przy odwołaniach między namespace'ami jest jednym z najczęstszych błędów przy pierwszym kontakcie z Kubernetesem — działa lokalnie w tej samej przestrzeni, przestaje działać, gdy usługę przenosimy albo gdy aplikacja próbuje odwołać się do usługi sąsiada.

ndots i wolne zapytania do domen zewnętrznych

> **Pojęcie: ndots.** Parametr resolwera DNS systemu Linux mówiący, ile kropek musi zawierać nazwa, żeby resolwer spróbował rozwiązać ją jako w pełni kwalifikowaną, zanim zacznie

> doklejać do niej domeny wyszukiwania z listy `search`. Kubernetes ustawia domyślnie `ndots:5` w każdym podzie.

Konsekwencja tej wartości jest konkretna: nazwa `api.zewnetrzny-dostawca.com` ma tylko dwie kropki, czyli mniej niż `ndots:5`, więc resolwer w pierwszej kolejności próbuje ją rozwiązać jako `api.zewnetrzny-dostawca.com.produkcja.svc.cluster.local`, potem `api.zewnetrzny-dostawca.com.svc.cluster.local`, potem `api.zewnetrzny-dostawca.com.cluster.local` — dopiero po tych trzech nieudanych próbach próbuje samej nazwy bez żadnego dopisku. Każda z tych prób to osobne zapytanie DNS z osobnym oczekiwaniem na odpowiedź (albo na timeout), więc pierwsze połączenie aplikacji do zewnętrznego API bywa zauważalnie wolniejsze niż to samo połączenie wykonane poza klastrem, mimo że docelowy serwer odpowiada natychmiast.


zapytanie:
api.zewnetrzny-dostawca.com
(2 kropki, mniej niz ndots:5)
1
proba 1:
...com.produkcja.svc.cluster.local
NXDOMAIN, 2
proba 2:
...com.svc.cluster.local
NXDOMAIN, 3
proba 3:
...com.cluster.local
NXDOMAIN, 4
proba 4:
api.zewnetrzny-dostawca.com
(bez dopisku)

Resolwer przy ndots 5 próbuje kolejno trzy domeny wyszukiwania i dopiero po trzech nieudanych próbach rozwiązuje nazwę zewnętrzną bez żadnego dopisku (kliknięcie otwiera pełny rozmiar)

Uważaj: dla aplikacji intensywnie korzystających z domen zewnętrznych rozwiązaniem bywa dopisanie kropki na końcu w pełni kwalifikowanej nazwy w kodzie aplikacji (`api.zewnetrzny-dostawca.com.`, z kropką na końcu) albo obniżenie `ndots` dla konkretnego poda przez pole `dnsConfig` — nie zmieniamy globalnie `ndots` dla całego klastra bez pełnego zrozumienia konsekwencji dla wszystkich pozostałych aplikacji.

Zmienne środowiskowe z adresami usług

> **Pojęcie: zmienne środowiskowe usług.** Starszy, poprzedzający CoreDNS mechanizm wykrywania usług — Kubernetes wstrzykuje do każdego poda zmienne środowiskowe typu `NAZWAUSLUGI_SERVICE_HOST` i `NAZWAUSLUGI_SERVICE_PORT` dla każdej usługi, która istniała w klastrze w momencie startu poda.

Mechanizm ten ma istotne ograniczenie, przez które dziś jest tylko ciekawostką historyczną: zmienne dostają wyłącznie pody uruchomione po utworzeniu danej usługi, bo są wstrzykiwane raz, przy starcie kontenera, i nigdy się nie odświeżają. Usługa utworzona już po starcie poda jest dla niego niewidoczna tą drogą, a jedynym sposobem na jej dostrzeżenie jest restart poda. DNS nie ma tego ograniczenia — rozwiązuje nazwę na bieżąco, przy każdym zapytaniu.

Diagnostyka awarii DNS w klastrze

Najczęstsze objawy awarii DNS i to, co sprawdzamy w pierwszej kolejności:

Objaw	Co sprawdzamy
każde połączenie z poda kończy się błędem `Temporary failure in name resolution`	czy pody CoreDNS w ogóle działają (`kubectl get pods -n kube-system -l k8s-app=kube-dns`)
rozwiązują się nazwy usług, nie rozwiązują się domeny zewnętrzne	reguły forward w `Corefile`, dostęp sieciowy z podów CoreDNS na zewnątrz, `NetworkPolicy` blokująca ruch wyjściowy opisana na stronie Sieć klastra i polityki sieciowe

Objaw	Co sprawdzamy
rozwiązywanie nazw działa, ale bardzo wolno	wartość `ndots`, obciążenie samych podów CoreDNS (`kubectl top pod -n kube-system`)
działa z jednego węzła, nie działa z innego	reguły `NetworkPolicy` albo CNI ograniczające ruch do adresu CoreDNS z konkretnego węzła, temat rozwinięty na stronie Sieć klastra i polityki sieciowe
nazwa usługi rozwiązuje się na stary adres po jej odtworzeniu	krótkotrwałe opóźnienie propagacji — sprawdzamy ponownie po kilku sekundach, zanim uznamy to za awarię

Częste błędy

Odwołanie do usługi w innej przestrzeni nazw bez podania przestrzeni. Aplikacja działa poprawnie w środowisku, gdzie wszystko stoi w jednej przestrzeni nazw, i zaczyna zwracać błędy połączenia natychmiast po rozdzieleniu usług na osobne przestrzenie — naprawą jest dopisanie przestrzeni nazw do adresu albo pełnej nazwy FQDN.

Twarde zakodowanie adresu ClusterIP w konfiguracji aplikacji. Adres ClusterIP potrafi się zmienić przy odtworzeniu usługi (np. po jej usunięciu i utworzeniu na nowo) — jedynym stabilnym punktem odniesienia jest nazwa, nigdy sam adres.

Poleganie na zmiennych środowiskowych usług przy usługach tworzonych dynamicznie. Pod uruchomiony przed daną usługą nigdy nie dostanie jej zmiennych środowiskowych bez restartu — DNS jest jedynym mechanizmem odpornym na kolejność startu.

□□ Ćwiczenia

> Ćwiczenia zakładają dostęp do klastra testowego z co najmniej dwiema przestrzeniami nazw i jedną usługą w każdej z nich.

Ćwiczenie 1 — porównanie krótkiej i pełnej nazwy

```
kubectl run diagnostyka --rm -it --image=busybox:1.36 --restart=Never -n NazwaPrzestrzeni --
sh -c "nslookup NazwaUslugi; nslookup NazwaUslugi.inna-przestrzen"
```

Uruchamiamy pod w jednej przestrzeni nazw i próbujemy rozwiązać nazwę usługi z tej samej przestrzeni krótką formą, a potem tę samą krótką formę wobec usługi w innej przestrzeni.

Odpowiedź

Krótka nazwa rozwiązuje się poprawnie tylko dla usługi we własnej przestrzeni nazw — dla usługi w innej przestrzeni krótka forma zwraca `NXDOMAIN`, dopóki nie dopiszemy nazwy przestrzeni.

Pytania kontrolne

1. Dlaczego pierwsze połączenie z poda do zewnętrznego API bywa wolniejsze niż kolejne?

Odpowiedź

Przy domyślnym `ndots:5` resolwer najpierw próbuje doklejać kolejne domeny wyszukiwania z listy `search` do nazwy zewnętrznej, zanim spróbuje jej bez żadnego dopisku — każda nieudana próba to osobne zapytanie DNS z czasem oczekiwania na odpowiedź.

2. Co się stanie z aplikacją, jeśli wszystkie pody CoreDNS ulegną awarii jednocześnie?

Odpowiedź

Żadna nowa nazwa się nie rozwiąże — połączenia korzystające z już otwartych gniazd i zapamiętanych adresów IP mogą jeszcze działać, ale każde nowe połączenie po nazwie (w tym do innych usług w klastrze) zacznie kończyć się błędem rozwiązywania nazwy.

Powiązane: Service: jak ruch trafia do poda · Sieć klastra i polityki sieciowe · Diagnostyka: objaw, przyczyna, pierwsze kroki

Źródła: dokumentacja kubernetes.io, rozdziały „DNS for Services and Pods" i „Debugging DNS Resolution" · dokumentacja coredns.io

---

Revision #33
Created 2026-09-25 10:33:02 UTC by Tomasz Konieczny
Updated 2026-10-26 14:17:04 UTC by Tomasz Konieczny