k3s w laboratorium

Lekki Kubernetes na k3s01: pierwszy deploy nginx w kolorach, skalowanie, aktualizacja i diagnostyka Traefika.

- k3s w laboratorium: pierwszy deploy, nginx w kolorach

k3s w laboratorium: pierwszy deploy, nginx w kolorach

◫◫ Poziom: średnie

◫◫ Dotyczy: k3s (kanał stable, aktualnie Kubernetes 1.36), nginx 1.30, Traefik wbudowany w k3s

▯ Czas: ~45 min

Stawiamy k3s na `k3s01` i wdrażamy najprostszy możliwy przykład: trzy pody zwykłego nginksa, z których każdy pokazuje stronę w innym kolorze i ze swoją nazwą. Dzięki temu każde odświeżenie przeglądarki albo każde kolejne wywołanie `curl` pokazuje gołym okiem, do którego poda akurat trafił ruch. Na tym samym przykładzie ćwiczymy skalowanie, samoleczenie, aktualizację bez przerwy i wycofanie zmian.

◫◫ Skróty użyte na tej stronie (5)

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
HTTP	Hypertext Transfer Protocol: protokół stron WWW
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)
TCP	Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia

Fiszka

◫◫ ZMIENIA SYSTEM

Instalacja k3s: curl | sh

```
curl -sfL https://get.k3s.io | sh -
```

Instalujemy k3s na maszynie `k3s01` jednym poleceniem: skrypt pobiera binarkę, instaluje ją jako usługę systemd i uruchamia serwer razem z wbudowanym Traefikiem, `containerd` i `local-path-provisioner`.

Przykład:

```
[INFO]  Finding release for channel stable
[INFO]  Using v1.36.4+k3s1 as release
[INFO]  Downloading hash https://github.com/k3s-
io/k3s/releases/download/v1.36.4+k3s1/sha256sum-amd64.txt
[INFO]  systemd: Starting k3s
```

□□ **curl** = client URL: pobieranie i wysyłanie przez HTTP z terminala

□□ ZMIENIA SYSTEM

Przygotowanie pliku kubeconfig do pracy bez roota

```
mkdir -p ~/.kube
sudo cat /etc/rancher/k3s/k3s.yaml > ~/.kube/config
chmod 600 ~/.kube/config
```

Kopiujemy plik dostępowy k3s do standardowej lokalizacji `kubectl`, żeby dalej pracować zwykłym `kubectl` zamiast `sudo k3s kubectl`.

□□ **mkdir** = make directory: utwórz katalog · **cat** = concatenate: sklej i wypisz pliki · **chmod** = change mode: zmień prawa dostępu

□□ TYLKO ODCZYT

Sprawdzenie stanu węzła: kubectl get nodes

```
kubectl get nodes
```

Przykład:

```
NAME    STATUS   ROLES                  AGE   VERSION
k3s01   Ready    control-plane,master   3m    v1.36.4+k3s1
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

ZMIENIA SYSTEM

Zastosowanie kompletu manifestów: kubectl apply -f

```
kubectl apply -f kolorowy-nginx.yaml
```

Jeden plik niżej zawiera cztery obiekty naraz, od `Namespace` po `Ingress`, więc jedno polecenie tworzy całe laboratorium.

Przykład:

```
namespace/kolory created
deployment.apps/kolorowy-nginx created
service/kolorowy-nginx created
ingress.networking.k8s.io/kolorowy-nginx created
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Sprawdzenie podów z adresami: kubectl get pods -o wide

```
kubectl get pods -n kolory -o wide
```

Przykład:

```
NAME                              READY   STATUS    RESTARTS   AGE   IP           NODE
kolorowy-nginx-6c8f9d7b5b-2xk4t   1/1     Running   0          40s   10.42.0.23   k3s01
kolorowy-nginx-6c8f9d7b5b-7mqzv   1/1     Running   0          40s   10.42.0.24   k3s01
kolorowy-nginx-6c8f9d7b5b-p9r2s   1/1     Running   0          40s   10.42.0.25   k3s01
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

ZMIENIA SYSTEM

Skalowanie wdrożenia: kubectl scale --replicas

```
kubectl scale deployment/kolorowy-nginx --replicas=5 -n kolory
```

Przykład:

```
deployment.apps/kolorowy-nginx scaled
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

OSTROŻNIE · NIEODWRACALNE

Usunięcie poda na żywo: kubectl delete pod

```
kubectl delete pod kolorowy-nginx-6c8f9d7b5b-2xk4t -n kolory
```

Usuwamy jeden konkretny pod, żeby zobaczyć, jak Deployment odtwarza go w miejsce usuniętego, z nową nazwą i nowym kolorem.

Przykład:

```
pod "kolorowy-nginx-6c8f9d7b5b-2xk4t" deleted
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

ZMIENIA SYSTEM

Podmiana obrazu w działającym wdrożeniu: kubectl set image

```
kubectl set image deployment/kolorowy-nginx nginx=nginx:1.29.9 -n kolory
```

Przykład:

```
deployment.apps/kolorowy-nginx image updated
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

ZMIENIA SYSTEM

Śledzenie postępu aktualizacji: kubectl rollout status

```
kubectl rollout status deployment/kolorowy-nginx -n kolory
```

Przykład:

```
Waiting for deployment "kolorowy-nginx" rollout to finish: 2 out of 5 new replicas have been updated...
deployment "kolorowy-nginx" successfully rolled out
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

ZMIENIA SYSTEM

Wycofanie aktualizacji: kubectl rollout undo

```
kubectl rollout undo deployment/kolorowy-nginx -n kolory
```

Przykład:

```
deployment.apps/kolorowy-nginx rolled back
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

OSTROŻNIE · NIEODWRACALNE

Sprzątanie: kubectl delete namespace

```
kubectl delete namespace kolory
```

Usuwamy naraz całą przestrzeń nazw i wszystko, co w niej stoi: Deployment, pody, Service i Ingress. Sam k3s na `k3s01` zostaje bez zmian, gotowy na kolejne laboratorium.

kubectl = Kubernetes control: główne narzędzie Kubernetesa

Z życia wzięte

Z ŻYCIA WZIĘTE

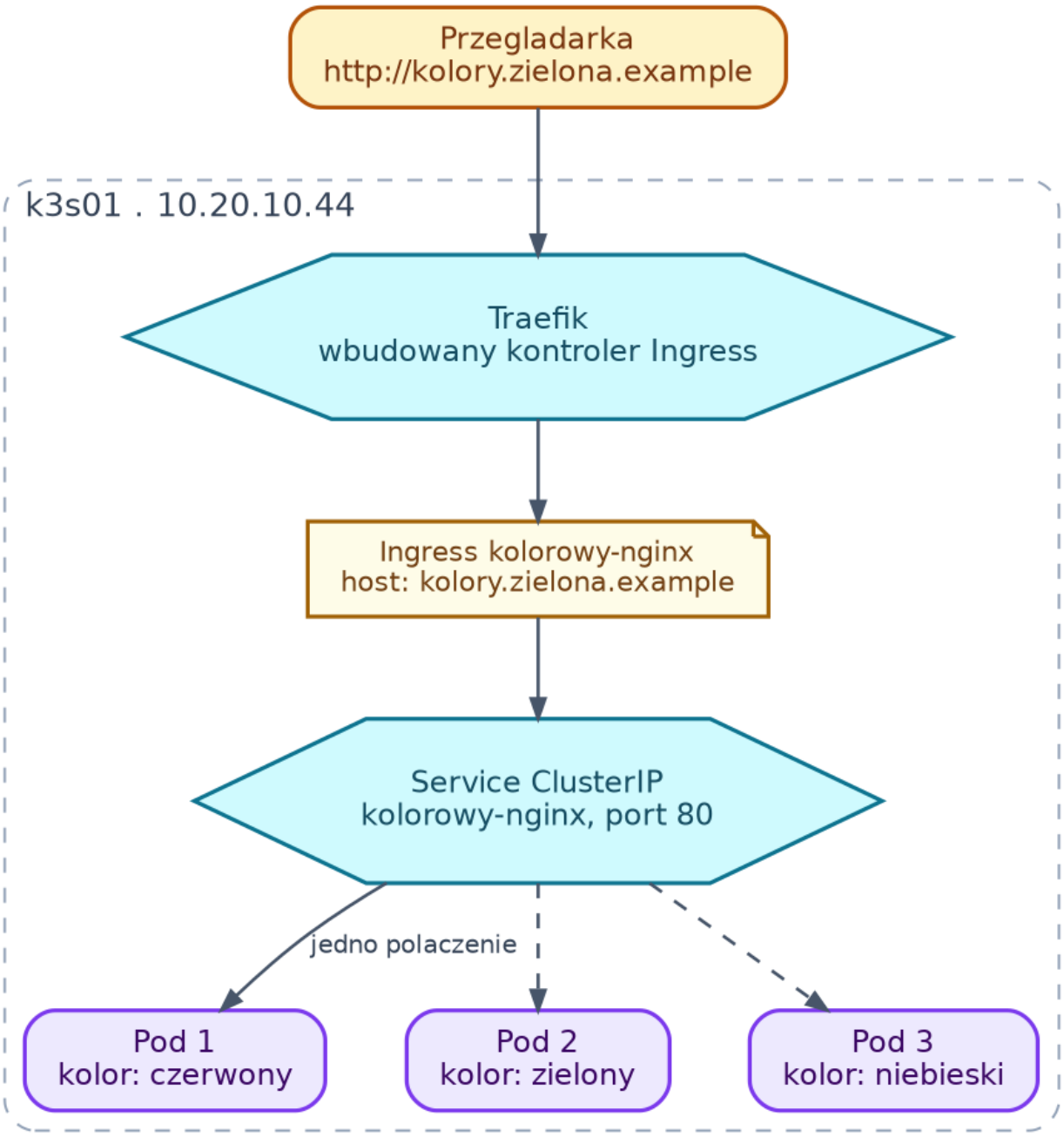
W sobotę po południu Tomek pokazuje panu Zbigniewowi na telefonie, co to znaczy „skalowanie". Otwiera `kolory.zielona.example` w przeglądarce, wykonuje w terminalu `kubectl scale deployment/kolorowy-nginx --replicas=5 -n kolory` i każe prezesowi odświeżać stronę. Po kilkunastu odświeżeniach pan Zbigniew widzi dwa nowe kolory, których wcześniej nie było, i pyta, czy to samo dzieje się z Optimą w piątek wieczorem. Tomek tłumaczy, że Optima stoi

> wciąż na jednej maszynie, ale właśnie dlatego ten kolorowy przykład jest w laboratorium, a nie na produkcji.

Jak to działa

Rozwiń: jak to działa pod spodem

Żądanie z przeglądarki albo z `curl` trafia po DNS na adres `k3s01`, gdzie nasłuchuje Traefik, wbudowany w k3s bez osobnej instalacji. Traefik dopasowuje regułę Ingress po hoście `kolory.zielona.example`, znajduje wskazany w niej Service `kolorowy-nginx` i kieruje ruch do jednego z trzech (albo pięciu, po skalowaniu) podów z listy Endpoints tego Service.



Droga żądania od przeglądarki przez Traefika i Service do jednego z trzech kolorowych podów nginksa (kliknięcie otwiera pełny rozmiar)

> **Pojęcie: Downward API.** Mechanizm, którym Kubernetes udostępnia kontenerowi informacje o samym sobie i o podzie, w którym działa, bez potrzeby pytania o nie `kube-apiserver` z zewnątrz. Najprostsza forma to zmienna środowiskowa wypełniona przez `fieldRef`, wskazujący pole z manifestu poda, na przykład `metadata.name` (nazwa poda)

albo `status.podIP` (adres IP poda, znany dopiero w chwili startu, nie wcześniej). Dokładnie tego mechanizmu używamy niżej, żeby każdy pod sam, bez naszej ingerencji, poznał własną nazwę i na jej podstawie wybrał kolor.

Wybór między `ConfigMap` a kontenerem inicjalizującym sprowadza się do tego, gdzie ma się dziać logika wyboru koloru. `ConfigMap` trzymałby gotowy plik `index.html` albo skrypt, ale sam kolor musiałby wtedy zależeć od czegoś stałego w konfiguracji, nie od nazwy poda znanej dopiero po starcie. Kontener inicjalizujący `generator-koloru` uruchamia się przed głównym kontenerem nginksa, odczytuje własną nazwę i adres IP przez Downward API, na ich podstawie liczy prosty skrót (`cksum`) i modulo z liczby kolorów wybiera jeden z sześciu, po czym zapisuje gotowy `index.html` do wspólnego wolumenu `emptyDir`, z którego czyta go już tylko nginx, bez żadnej własnej logiki.

```
apiVersion: v1
kind: Namespace
metadata:
  name: kolory
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kolorowy-nginx
  namespace: kolory
  labels:
    app: kolorowy-nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: kolorowy-nginx
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    metadata:
      labels:
        app: kolorowy-nginx
```

```
    spec:
      initContainers:
        - name: generator-koloru
          image: busybox:1.37.0
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: POD_IP
              valueFrom:
                fieldRef:
                  fieldPath: status.podIP
          command:
            - sh
            - -c
            - |
              set -e
              KOLORY="czerwony:#ef4444 zielony:#22c55e niebieski:#3b82f6 zolty:#eab308
fioletowy:#a855f7 pomaranczowy:#f97316"
              LICZBA=$(echo $KOLORY | wc -w)
              HASH=$(echo -n "$POD_NAME" | cksum | cut -d' ' -f1)
              INDEKS=$((HASH % LICZBA))
              WYBRANY=$(echo $KOLORY | cut -d' ' -f$((INDEKS + 1)))
              NAZWA=$(echo $WYBRANY | cut -d':' -f1)
              HEX=$(echo $WYBRANY | cut -d':' -f2)
              echo "<html><body style='background-color:$HEX; font-family:sans-serif;
text-align:center; padding-top:15%;'>" > /strona/index.html
              echo "<h1>Pod: $POD_NAME</h1>" >> /strona/index.html
              echo "<h2>Kolor: $NAZWA</h2>" >> /strona/index.html
              echo "<h3>Adres IP: $POD_IP</h3>" >> /strona/index.html
              echo "</body></html>" >> /strona/index.html
          volumeMounts:
            - name: strona
              mountPath: /strona
      containers:
        - name: nginx
          image: nginx:1.30.5
```

```
          ports:
            - containerPort: 80
          readinessProbe:
            httpGet:
              path: /
              port: 80
            periodSeconds: 5
            failureThreshold: 3
          livenessProbe:
            httpGet:
              path: /
              port: 80
            periodSeconds: 10
            failureThreshold: 3
          resources:
            requests:
              cpu: "50m"
              memory: "32Mi"
            limits:
              cpu: "200m"
              memory: "64Mi"
          volumeMounts:
            - name: strona
              mountPath: /usr/share/nginx/html
              readOnly: true
      volumes:
        - name: strona
          emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
  name: kolorowy-nginx
  namespace: kolory
spec:
  type: ClusterIP
  selector:
    app: kolorowy-nginx
```

```
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: kolorowy-nginx
  namespace: kolory
spec:
  ingressClassName: traefik
  rules:
    - host: kolory.zielona.example
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: kolorowy-nginx
                port:
                  number: 80
```

Kontener nginxsa mocuje ten sam wolumen `strona` w trybie tylko do odczytu: sam nie ma żadnej logiki, wyłącznie serwuje plik, który wcześniej zostawił dla niego kontener inicjalizujący. Sondy gotowości i żywotności pytają o zwykłą stronę główną: skoro nginx nie łączy się z żadną bazą ani zewnętrzną usługą, sama odpowiedź HTTP 200 wystarcza za dowód, że kontener działa poprawnie.

Weryfikacja

```
kubectl get pods -n kolory -o wide
```

Sprawdzamy, że wszystkie pody mają status `Running`, `1/1 Ready`, i każdy inny adres IP z puli `10.42.0.0/16` przydzielonej przez `local-path-provisioner` i sieć wbudowaną w k3s.

```
for i in $(seq 1 6); do
  curl -s --resolve kolory.zielona.example:80:10.20.10.44 http://kolory.zielona.example/ |
grep "Kolor:"
done
```

Przykład:

```
<h2>Kolor: zielony</h2>
<h2>Kolor: niebieski</h2>
<h2>Kolor: zielony</h2>
<h2>Kolor: czerwony</h2>
<h2>Kolor: niebieski</h2>
<h2>Kolor: zielony</h2>
```

Sześć wywołań w pętli daje wymieszane kolory, bo każde osobne wywołanie `curl` otwiera nowe połączenie TCP, a `kube-proxy` przy każdym nowym połączeniu wybiera jeden z żywych podów z listy Endpoints, mniej więcej równomiernie. Przeglądarka zachowuje się inaczej: protokół HTTP/1.1 domyślnie trzyma połączenie otwarte (`keep-alive`), więc kolejne żądania tej samej karty do tego samego hosta wracają tym samym, już otwartym połączeniem do tego samego poda. Samo wciśnięcie F5 może więc pokazywać ciągle ten sam kolor, mimo że za usługą stoją trzy różne pody. Żeby zobaczyć zmianę w przeglądarce, otwieramy nową kartę albo okno incognito zamiast odświeżać starą, a żeby udowodnić, że serwer naprawdę rozkłada ruch, wracamy do pętli `curl` z osobnymi połączeniami, tak jak wyżej.

□□ Ćwiczenia

Skalowanie: kubectl scale --replicas

```
kubectl scale deployment/kolorowy-nginx --replicas=5 -n kolory
kubectl get pods -n kolory -o wide
```

Dwa nowe pody startują z dwiema nowymi, losowymi nazwami, a każda nowa nazwa daje inny wynik `cksum`, więc widzimy kolory, których chwilę wcześniej w ogóle nie było w puli.

Samoleczenie: kubectl delete pod

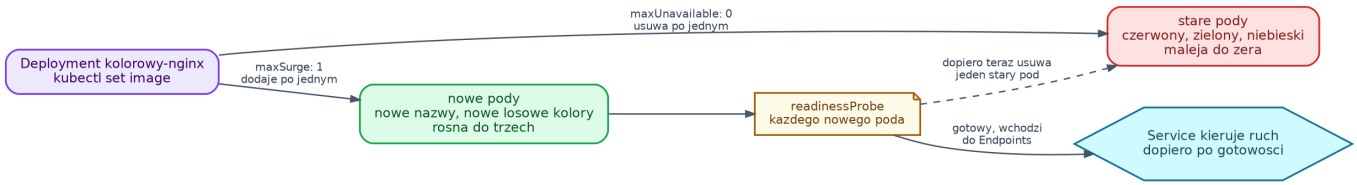
```
kubectl delete pod NazwaJednegoZPodow -n kolory
kubectl get pods -n kolory -o wide
```

ReplicaSet natychmiast tworzy nowy pod w miejsce usuniętego, z nową nazwą, nowym adresem IP i, co najciekawsze w tym przykładzie, zwykle innym kolorem niż miał usunięty pod.

Aktualizacja bez przerwy: kubectl set image i rollout status

```
kubectl set image deployment/kolorowy-nginx nginx=nginx:1.29.9 -n kolory
kubectl rollout status deployment/kolorowy-nginx -n kolory
```

Zmiana wersji obrazu nginksa niczego nie zmienia w samej logice kolorów, ale wymusza nowy szablon poda, więc Deployment tworzy nowy ReplicaSet i przechodzi przez dokładnie tę samą sekwencję `RollingUpdate` co każda inna aktualizacja w tej książce, tyle że tutaj każdy krok jest widoczny gołym okiem jako zmiana koloru.



Deployment zwiększa nowy zestaw podów o nowych kolorach i zmniejsza stary, a Service kieruje ruch do nowego poda dopiero po jego gotowości (kliknięcie otwiera pełny rozmiar)

Ten sam efekt da się wywołać bez zmiany wersji obrazu: wystarczy dopisać kolejny wpis do listy `KOLORY` w kontenerze inicjalizującym i wykonać `kubectl apply -f kolorowy-nginx.yaml` ponownie, bo to wciąż zmiana `spec.template`, uruchamiająca ten sam mechanizm.

Wycofanie: kubectl rollout undo

```
kubectl rollout undo deployment/kolorowy-nginx -n kolory
kubectl rollout status deployment/kolorowy-nginx -n kolory
```

Deployment wraca do poprzedniego ReplicaSetu, zachowanego z liczbą replik zerową właśnie na taką okazję, tą samą sekwencją `RollingUpdate`, tylko w przeciwnym kierunku.

Gdy Traefik odpowiada 502

> Kolejność kroków jest ta sama niezależnie od tego, jaka aplikacja stoi za Ingressem: prosty, kolorowy nginx ułatwia naukę tej kolejności właśnie dlatego, że wyklucza z góry najbardziej mylącą przyczynę, czyli błąd w samej aplikacji.

1. Reguła Ingress: host i ścieżka

```
kubectl describe ingress kolorowy-nginx -n kolory
```

Sprawdzamy, czy pole `Rules` zawiera dokładnie `kolory.zielona.example` i czy `IngressClass` wskazuje na `traefik`.

2. Service: czy istnieje i na jaki port wskazuje

```
kubectl get svc kolorowy-nginx -n kolory -o wide
```

Numer `port` z tej usługi musi się zgadzać z polem `service.port.number` w regule Ingress.

3. Endpoints i EndpointSlices: czy jest komu odpowiedzieć

```
kubectl get endpoints kolorowy-nginx -n kolory
kubectl get endpointslices -n kolory -l kubernetes.io/service-name=kolorowy-nginx
```

Pusta lista adresów w obu komendach oznacza, że selektor Service nie pasuje do etykiety `app: kolorowy-nginx` na podach albo żaden pod nie przeszedł sondy gotowości. Traefik odpowiada wtedy zwykle 404 (albo 503 przy włączonym `allowEmptyServices`), a nie 502; 502 dostajemy, gdy adres poda jest znany, ale połączenie z nim się nie udaje (zły `targetPort`, aplikacja słucha tylko na 127.0.0.1, zła nazwa portu). Pełny tok diagnozy z przykładu Camundy opisuje Częste problemy z silnikami procesów: Traefik 502, Zeebe, PVC i OIDC.

4. targetPort kontenera

```
kubectl get pod -n kolory -o jsonpath='{.items[*].spec.containers[*].ports}'
```

Sprawdzamy, czy nginx w kontenerze faktycznie nasłuchuje na porcie 80, dokładnie tym, który Service wskazuje jako `targetPort`.

5. Port forward z pominięciem Traefika

```
kubectl port-forward svc/kolorowy-nginx 8080:80 -n kolory
```

Jeśli `http://localhost:8080` pokazuje kolorową stronę, wykluczamy nginksa i Service jako przyczynę: problem leży między Traefikiem a Service.

6. Logi Traefika

```
kubectl logs -n kube-system -l app.kubernetes.io/name=traefik --tail=100
```

Wbudowany Traefik loguje do tego samego miejsca co każdy inny pod w klastrze, więc `kubectl logs` na jego etykiecie pokazuje próby połączenia z backendem i dokładny powód każdego odrzucenia.

Częste błędy

Objaw	Przyczyna	Co robimy
`kubectl get ingress` pokazuje regułę, ale pole `ADDRESS` jest puste	zła `ingressClassName` albo Traefik jeszcze nie wystartował	`kubectl get pods -n kube-system -l app.kubernetes.io/name=traefik`
502 na każdej ścieżce	zły `targetPort` albo aplikacja nie słucha na adresie poda (puste Endpoints dają raczej 404 lub 503)	krok 3 i 4 listy kontrolnej wyżej
każdy pod pokazuje ten sam kolor	wszystkie pody dostały tę samą, powieloną nazwę w teście lokalnym poza Deploymentem	tworzymy pody wyłącznie przez Deployment, nigdy ręcznie z tą samą nazwą
przeglądarka ciągle pokazuje jeden kolor mimo skalowania	otwarte połączenie `keep-alive` do tego samego poda	nowa karta, tryb incognito albo pętla `curl` z osobnymi połączeniami
nginx zwraca `403 Forbidden` zamiast strony	kontener inicjalizujący nie zdążył zapisać `index.html` przed startem nginksa	sprawdzamy kolejność kontenerów przez `kubectl describe pod`, init musi zakończyć się kodem 0 przed startem głównego kontenera

□□ Ćwiczenia

>

> Zadania sprawdzające.

1. Wyjaśniamy, dlaczego sam kontener nginksa w tym przykładzie nie potrzebuje żadnej własnej logiki wyboru koloru.

Odpowiedź

Kolor wybiera i zapisuje do pliku `index.html` kontener inicjalizujący, uruchamiany raz, przed startem głównego kontenera, a nginx tylko odczytuje gotowy plik ze wspólnego wolumenu `emptyDir`, w trybie tylko do odczytu.

1. Po `kubectl scale --replicas=5` liczba widocznych kolorów rośnie, mimo że lista możliwych kolorów w skrypcie się nie zmieniła. Dlaczego?

Odpowiedź

Każdy nowy pod dostaje nową, losowo wygenerowaną nazwę, a kolor liczony jest ze skrótu (`cksum`) tej właśnie nazwy modulo liczba kolorów na liście, więc nowa nazwa może trafić na kolor, którego wśród już istniejących podów jeszcze nie było.

1. `curl` w pętli pokazuje kilka różnych kolorów, ale odświeżanie tej samej karty w przeglądarce wciąż pokazuje jeden. Co to oznacza i co sprawdzamy w pierwszej kolejności?

Odpowiedź

To nie usterka: przeglądarka trzyma jedno połączenie HTTP z `keep-alive` otwarte do tego samego poda, więc kolejne żądania tej samej karty wracają tą samą drogą. Sprawdzamy to, otwierając nową kartę albo okno incognito, zamiast szukać przyczyny w Service czy Ingressie.

1. `kubectl get endpoints kolorowy-nginx -n kolory` pokazuje pustą listę, mimo że `kubectl get pods -n kolory` pokazuje trzy pody w stanie `Running`. Co sprawdzamy najpierw?

Odpowiedź

Zgodność selektora Service (`kubectl get svc kolorowy-nginx -n kolory -o yaml`, pole `spec.selector`) z etykietami podów (`kubectl get pod -n kolory --show-labels`), a potem stan sondy gotowości w `kubectl describe pod`: pod `Running`, który jeszcze nie przeszedł sondy gotowości, nie trafia na listę Endpoints.

Powiązane: Kubernetes: sternik, który przypłynął z Google · Pod: najmniejsza jednostka · Deployment, ReplicaSet i aktualizacje bez przerwy · Service: jak ruch trafia do poda · Ingress i Gateway API: wejście z zewnątrz

Źródła: docs.k3s.io, rozdziały „Installation" i „Networking" (Traefik, ServiceLB, local-path-provisioner) · kubernetes.io/docs/concepts/workloads/pods/downward-api/ · kubernetes.io/docs/concepts/workloads/controllers/deployment/ · hub.docker.com/_/nginx, tagi wersji stabilnej · github.com/k3s-io/k3s/releases