kubectl: narzędzie codzienne

Poziom: podstawy **Dotyczy: kubectl 1.37, Kubernetes 1.35–1.37 (skew ±1 wersja)**

Czas: ~35 min

Komendy wykonujemy z CLI bash jako user mający dostęp do klastra przez skonfigurowany `~/.kube/config` — bez poprawnego kontekstu każde polecenie kończy się błędem połączenia albo `Unauthorized`.

Skróty użyte na tej stronie (5)

Skrót	Co znaczy
CA	Certificate Authority: urząd certyfikacji, który podpisuje certyfikaty
CLI	Command Line Interface: praca przez wpisywanie komend w terminalu
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)
RBAC	Role-Based Access Control: uprawnienia nadawane rolom, a role ludziom
YAML	YAML Ain't Markup Language: format plików konfiguracyjnych oparty na wcięciach

Fiszka

TYLKO ODCZYT

Wylistowanie dostępnych kontekstów: kubectl config get-contexts

```
kubectl config get-contexts
```

Przykład:

```
CURRENT   NAME            CLUSTER      AUTHINFO        NAMESPACE
*         produkcja       k8s-prod     admin-prod      produkcja
          test            k8s-test     admin-test      default
          lokalny-kind    kind-dev     kind-dev        default
```

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

□□ TYLKO ODCZYT

Przełączenie aktywnego kontekstu: kubectl config use-context

```
kubectl config use-context test
```

Przykład:

```
Switched to context "test".
```

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

□□ ZMIENIA SYSTEM

Ustawienie domyślnej przestrzeni nazw dla bieżącego kontekstu: kubectl config set-context --current

```
kubectl config set-context --current --namespace=produkcja
```

Zmieniamy przestrzeń nazw zapisaną w bieżącym kontekście, żeby nie dopisywać `-n produkcja` do każdej kolejnej komendy w tej sesji terminala.

Przykład:

```
Context "produkcja" modified.
```

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

□□ TYLKO ODCZYT

Wylistowanie zasobów z dodatkowymi kolumnami: kubectl get -o wide

```
kubectl get pods -o wide
```

Dokładamy do standardowego wyniku adres IP poda, węzeł, na którym działa, oraz nazwę węzła nominowanego — informacje pomijane w domyślnym, wąskim widoku.

Przykład:

```
NAME                              READY   STATUS    RESTARTS   AGE   IP
NODE    NOMINATED NODE   READINESS GATES
katalog-produktow-6f9b7c9-4kxpl   1/1     Running   0          3h    10.244.1.12
k3s02   <none>           <none>
katalog-produktow-6f9b7c9-9wq2z   1/1     Running   0          3h    10.244.2.8
k3s03   <none>           <none>
```

> Nazwy węzłów w wynikach (`k3s01`, `k3s02`, `k3s03`, adresy 10.20.10.44, 10.20.10.67 i 10.20.10.68) pochodzą z laboratorium k3s Zielonej, bo na nim ćwiczymy `kubectl`; budowę klastra przez `kubeadm` opisujemy osobno jako sposób ogólny.

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

TYLKO ODCZYT

Pobranie pełnej definicji obiektu w formacie YAML: kubectl get -o yaml

```
kubectl get deployment katalog-produktow -o yaml
```

Zwracamy kompletny obiekt tak, jak faktycznie leży w `etcd` — łącznie z polami wypełnionymi domyślnie przez klaster, których nie było w naszym manifeście. Przydatne, gdy chcemy zobaczyć, co Kubernetes dopisał sam, albo skopiować istniejący obiekt jako punkt wyjścia do nowego manifestu.

Przykład:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: katalog-produktow
  namespace: produkcja
  generation: 4
  resourceVersion: "88213"
spec:
  replicas: 3
  selector:
    matchLabels:
```

```
      app: katalog-produktow
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
status:
  replicas: 3
  readyReplicas: 3
  availableReplicas: 3
```

YAML = YAML Ain't Markup Language: format plików konfiguracyjnych oparty na wcięciach · **etcd** = etc distributed: rozproszony magazyn par klucz-wartość, w którym Kubernetes trzyma cały stan klastra, a zmiany uzgadnia protokołem Raft · **kubectl** = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Wyciągnięcie pojedynczego pola: kubectl get -o jsonpath

```
kubectl get pods -l app=katalog-produktow -o jsonpath='{.items[*].status.podIP}'
```

Wyciągamy z wyniku dokładnie tę wartość, która nas interesuje, pomijając resztę — nieocenione w skryptach, gdzie dalej chcemy przetwarzać samą wartość, a nie sparsowany YAML.

Przykład:

```
10.244.1.12 10.244.2.8 10.244.1.15
```

YAML = YAML Ain't Markup Language: format plików konfiguracyjnych oparty na wcięciach · **kubectl** = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Śledzenie zmian zasobów na bieżąco: kubectl get --watch

```
kubectl get pods --watch
```

Zamiast jednorazowego zrzutu stanu, `kubectl` utrzymuje otwarte połączenie i dopisuje nową linię za każdym razem, gdy stan któregoś poda się zmienia — wygodne przy obserwowaniu

przebiegu wdrożenia albo skalowania na żywo, bez odpytywania w pętli.

Przykład:

```
NAME                               READY   STATUS    RESTARTS   AGE
katalog-produktow-6f9b7c9-4kxpl    1/1     Running   0          3h
katalog-produktow-7d8f6b5-xk2mn    0/1     Pending   0          0s
katalog-produktow-7d8f6b5-xk2mn    0/1     ContainerCreating   0   1s
katalog-produktow-7d8f6b5-xk2mn    1/1     Running   0          4s
katalog-produktow-6f9b7c9-4kxpl    1/1     Terminating   0        3h
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Szczegółowy opis obiektu ze zdarzeniami: kubectl describe

```
kubectl describe pod katalog-produktow-6f9b7c9-4kxpl
```

W przeciwieństwie do `kubectl get`, `describe` dokłada sekcję `Events` na końcu — chronologiczny dziennik tego, co płaszczyzna sterowania i `kubelet` faktycznie zrobiły z tym obiektem. To pierwsze miejsce do sprawdzenia, gdy pod nie startuje, a `kubectl get` pokazuje tylko suchy status bez przyczyny.

Przykład:

```
Name:         katalog-produktow-6f9b7c9-4kxpl
Namespace:    produkcja
Node:         k3s02/10.20.10.67
Status:       Running
IP:           10.244.1.12
Containers:
  katalog-produktow:
    Image:         git.zielona.example/zielona/katalog-produktow:2.4.1
    State:          Running
      Started:      Wed, 23 Sep 2026 08:12:04 +0200
    Ready:          True
    Restart Count:  0

Events:
```

```
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  3h    default-scheduler  Successfully assigned produkcja/katalog-
produktow-6f9b7c9-4kxpl to k3s02
  Normal  Pulled     3h    kubelet            Container image
"git.zielona.example/zielona/katalog-produktow:2.4.1" already present on machine
  Normal  Created    3h    kubelet            Created container katalog-produktow
  Normal  Started    3h    kubelet            Started container katalog-produktow
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Podgląd logów kontenera na bieżąco: kubectl logs -f

```
kubectl logs -f katalog-produktow-6f9b7c9-4kxpl
```

Przykład:

```
2026-09-23T08:12:05.014Z INFO  Server started on port 8080
2026-09-23T08:12:41.220Z INFO  GET /api/produkty 200 12ms
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Podgląd logów poprzedniej instancji kontenera po restarcie: kubectl logs --previous

```
kubectl logs --previous katalog-produktow-6f9b7c9-4kxpl
```

Kontener, który padł i został zrestartowany przez `kubelet`, dostaje nowy proces, a jego stare logi normalnie zniknęłyby razem z nim — `--previous` pokazuje logi z tej poprzedniej, już nieistniejącej instancji, co jest jedynym sposobem zobaczenia, co doprowadziło do awarii, zamiast oglądać logi już czystego, świeżo wystartowanego procesu.

Przykład:

```
2026-09-23T07:58:12.501Z ERROR Out of memory: heap limit exceeded
2026-09-23T07:58:12.502Z FATAL Process terminated
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Ograniczenie logów do ostatniego okresu: kubectl logs --since

```
kubectl logs --since=10m katalog-produktow-6f9b7c9-4kxpl
```

Przykład:

```
2026-09-23T08:05:12.100Z INFO  GET /api/produkty 200 9ms
2026-09-23T08:09:47.331Z INFO  GET /api/produkty 200 11ms
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Ograniczenie logów do ostatnich N linii: kubectl logs --tail

```
kubectl logs --tail=50 katalog-produktow-6f9b7c9-4kxpl
```

Zwracamy wyłącznie ostatnie 50 linii zamiast pełnej historii od startu kontenera — praktyczne przy długo działających kontenerach, których pełne logi liczą się w gigabajtach.

kubectl = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Wejście do powłoki działającego kontenera: kubectl exec -it

```
kubectl exec -it katalog-produktow-6f9b7c9-4kxpl -- bash
```

Odpowiednik `docker exec -it` (patrz strona Codzienne komendy), tylko przez `kube-apiserver` jako pośrednika — `kubectl` nie łączy się bezpośrednio z węzłem, żądanie przechodzi przez tę samą warstwę uwierzytelniania i autoryzacji, co każde inne polecenie `kubectl`.

Przykład:

```
root@katalog-produktow-6f9b7c9-4kxpl:/app#
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

□□ ZMIENIA SYSTEM

Zastosowanie manifestu deklaratywnie: kubectl apply -f

```
kubectl apply -f wdrozenie.yaml
```

Przykład:

```
deployment.apps/katalog-produktow configured
```

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

□□ TYLKO ODCZYT

Podgląd różnicy przed zastosowaniem: kubectl diff -f

```
kubectl diff -f wdrozenie.yaml
```

Pokazujemy dokładnie to, co `apply` by zmieniło, zanim faktycznie to zrobimy — bezcenne przed wprowadzeniem zmiany na produkcji, żeby nie odkryć niespodziewanej różnicy dopiero po fakcie.

Przykład:

```
diff -u -N /tmp/LIVE-123456789/apps.v1.Deployment.produkcja.katalog-produktow
/tmp/MERGED-123456789/apps.v1.Deployment.produkcja.katalog-produktow
--- /tmp/LIVE-123456789/apps.v1.Deployment.produkcja.katalog-produktow
+++ /tmp/MERGED-123456789/apps.v1.Deployment.produkcja.katalog-produktow
@@ -14,7 +14,7 @@
   template:
     spec:
       containers:
-      - image: git.zielona.example/zielona/katalog-produktow:2.4.0
+      - image: git.zielona.example/zielona/katalog-produktow:2.4.1
```

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

□□ OSTROŻNIE · NIEODWRACALNE

Usunięcie obiektu: kubectl delete

```
kubectl delete -f wdrozenie.yaml
```

Przykład:

```
deployment.apps "katalog-produktow" deleted
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

ZMIENIA SYSTEM

Sprawdzenie postępu wdrożenia: kubectl rollout status

```
kubectl rollout status deployment/katalog-produktow
```

Przykład:

```
Waiting for deployment "katalog-produktow" rollout to finish: 2 out of 3 new replicas
have been updated...
deployment "katalog-produktow" successfully rolled out
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

ZMIENIA SYSTEM

Cofnięcie wdrożenia do poprzedniej wersji: kubectl rollout undo

```
kubectl rollout undo deployment/katalog-produktow
```

Przykład:

```
deployment.apps/katalog-produktow rolled back
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

ZMIENIA SYSTEM

Ręczne przeskalowanie: kubectl scale

```
kubectl scale deployment/katalog-produktow --replicas=5
```

Przykład:

```
deployment.apps/katalog-produktow scaled
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Przekierowanie portu z poda na maszynę lokalną: kubectl port-forward

```
kubectl port-forward pod/katalog-produktow-6f9b7c9-4kxpl 8080:8080
```

Otwieramy tunel przez `kube-apiserver` do konkretnego poda, dostępny lokalnie pod `localhost:8080` — wygodne przy diagnostyce usługi, która normalnie nie jest wystawiona na zewnątrz klastra, bez zmieniania niczego w konfiguracji sieciowej.

Przykład:

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

kubectl = Kubernetes control: główne narzędzie Kubernetesa

ZMIENIA SYSTEM

Skopiowanie pliku między podem a hostem: kubectl cp

```
kubectl cp produkcja/katalog-produktow-6f9b7c9-4kxpl:/app/logs/error.log ./error.log
```

Przykład:

```
tar: Removing leading '/' from member names
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Podgląd zużycia zasobów przez pody: kubectl top

```
kubectl top pods
```

Wymaga zainstalowanego `metrics-server` w klastrze — bez niego komenda kończy się błędem `error: Metrics API not available`.

Przykład:

```
NAME                              CPU(cores)   MEMORY(bytes)
katalog-produktow-6f9b7c9-4kxpl   12m          186Mi
katalog-produktow-6f9b7c9-9wq2z   9m           179Mi
```

□□ **API** = Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą · **kubectl** = Kubernetes control: główne narzędzie Kubernetesa

□□ TYLKO ODCZYT

Wylistowanie zdarzeń klastra posortowanych czasem: kubectl get events --sort-by

```
kubectl get events --sort-by='.lastTimestamp' -n produkcja
```

Zdarzenia znikają domyślnie po godzinie, więc to polecenie ma sens uruchamiać zaraz po zaobserwowaniu problemu, nie następnego dnia.

Przykład:

```
LAST SEEN   TYPE      REASON             OBJECT
MESSAGE
2m          Warning   FailedScheduling   pod/katalog-produktow-7d8f6b5-xk2mn     0/3
nodes are available: 3 Insufficient memory.
45s         Normal    Scheduled          pod/katalog-produktow-7d8f6b5-xk2mn
Successfully assigned produkcja/katalog-produktow-7d8f6b5-xk2mn to k3s03
```

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

□□ TYLKO ODCZYT

Sprawdzenie, czy mamy uprawnienia do wykonania akcji: kubectl auth can-i

```
kubectl auth can-i delete deployments -n produkcja
```

Sprawdzamy uprawnienia RBAC bieżącego użytkownika bez faktycznego wykonywania akcji — przydatne przed skryptem automatyzującym coś na produkcji, żeby błąd braku uprawnień wykrył sam skrypt na starcie, a nie w połowie sekwencji.

Przykład:

```
yes
```

RBAC = Role-Based Access Control: uprawnienia nadawane rolom, a role ludziom · **kubectl** = Kubernetes control: główne narzędzie Kubernetesa

ZMIENIA SYSTEM

Wygenerowanie szkieletu manifestu bez wysyłania go do klastra: --dry-run=client -o yaml

```
kubectl create deployment katalog-produktow --image=git.zielona.example/zielona/katalog-produktow:2.4.1 --dry-run=client -o yaml
```

Przykład:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: katalog-produktow
  name: katalog-produktow
spec:
  replicas: 1
  selector:
    matchLabels:
      app: katalog-produktow
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
```

```
      labels:
        app: katalog-produktow
    spec:
      containers:
      - image: git.zielona.example/zielona/katalog-produktow:2.4.1
        name: katalog-produktow
status: {}
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

Z życia wzięte

Z ŻYCIA WZIĘTE

O 16:30 Tomek wykonuje `kubectl delete -f wdrozenie.yaml`, żeby posprzątać po teście, i dopiero po komunikacie `deployment.apps "katalog-produktow" deleted` zauważa, że gwiazdka w `kubectl config get-contexts` stała przy kontekście `produkcja`, nie `test`. Przez chwilę usługa katalogu produktów jest niedostępna, zanim `kubectl apply -f wdrozenie.yaml` z tego samego pliku w repozytorium odtwarza Deployment i wszystkie trzy pody wracają do stanu `Running`. Od tego dnia Tomek zaczyna każdą sesję pracy na klastrze od sprawdzenia, który kontekst ma gwiazdkę.

Plik kubeconfig — klaster, użytkownik, kontekst

Pojęcie: kubeconfig. Plik YAML, domyślnie `~/.kube/config`, w którym `kubectl` trzyma wszystko, co potrzebne do połączenia się z jednym albo wieloma klastrami. Składa się z trzech osobnych list, które dopiero kontekst łączy w jedną całość:

```
apiVersion: v1
kind: Config
clusters:
  - name: k8s-prod
    cluster:
```

```
      server: https://10.20.10.44:6443
      certificate-authority-data: LS0tLS1CRUdJTi...
users:
  - name: admin-prod
    user:
      client-certificate-data: LS0tLS1CRUdJTi...
      client-key-data: LS0tLS1CRUdJTi...
contexts:
  - name: produkcja
    context:
      cluster: k8s-prod
      user: admin-prod
      namespace: produkcja
current-context: produkcja
```

Klaster to sam adres `kube-apiserver` i certyfikat CA potrzebny, żeby `kubectl` mógł zweryfikować, że łączy się z właściwym serwerem, a nie z podstawionym. **Użytkownik** to poświadczenia — certyfikat klienta, token albo wywołanie zewnętrznego programu uwierzytelniającego (np. wtyczki chmurowej). **Kontekst** łączy jeden klaster, jednego użytkownika i domyślną przestrzeń nazw w jeden nazwany zestaw, między którymi przełączamy się poleceniem `kubectl config use-context` zamiast ręcznie edytować plik za każdym razem.

Jedna typowa stacja robocza administratora ma w `kubeconfig` kilka kontekstów naraz — produkcję, środowisko testowe i lokalny klaster do eksperymentów postawiony narzędziem `kind` albo `minikube` (oba są aktualnie oficjalnie rekomendowanymi sposobami uruchomienia Kubernetesa na własnej maszynie do nauki i developmentu, bez ryzyka pomyłkowego wykonania czegoś na realnym klastrze). Zanim wykonamy cokolwiek destrukcyjnego, dobrym nawykiem jest sprawdzenie `kubectl config get-contexts` i upewnienie się, który kontekst ma gwiazdkę przy `CURRENT`.

Tryb deklaratywny kontra imperatywny w kubectl

`kubectl` udostępnia oba podejścia opisane na stronie Po co Kubernetes, skoro mamy Dockera i to od nas zależy, którego użyjemy w danej sytuacji.

Polecenia imperatywne (`kubectl run`, `kubectl create deployment`, `kubectl scale`, `kubectl set image`) mówią klastrowi wprost, co zrobić, w tej chwili, i nie zostawiają po sobie żadnego pliku opisującego wynik. `kubectl run` tworzy dokładnie jeden pod i od wydania Kubernetes 1.18 nie potrafi już nic więcej — wcześniejsze mechanizmy generatorów, które pozwalały tą samą komendą

stworzyć też Deployment czy Job, zostały usunięte. Traktujemy go jako narzędzie do szybkiego testu albo debugowania, nie jako sposób wdrażania czegokolwiek na stałe.

`kubectl apply -f` działa deklaratywnie — porównuje trzy stany naraz: to, co jest zapisane w pliku teraz, to, co było zapisane przy poprzednim `apply` (zachowane w adnotacji `last-applied-configuration`, patrz strona Obiekty, manifesty i przestrzenie nazw), i to, co faktycznie jest aktualnie w klastrze — po czym oblicza trójstronny scalony wynik (three-way merge) i wysyła tylko różnicę. Dzięki temu `apply` nie nadpisuje pól, które ktoś (albo jakiś kontroler) zmienił ręcznie poza naszym manifestem, o ile sami tych pól nie dotykamy.

Na produkcji trzymamy się wyłącznie `apply` na plikach trzymanych w repozytorium, z dwóch powodów naraz. Po pierwsze manifest w repozytorium jest źródłem prawdy, które da się przejrzeć w code review, zanim trafi do klastra — polecenie imperatywne wpisane ręcznie w terminalu nie zostawia takiego śladu. Po drugie `apply` jest idempotentne — wykonanie tej samej komendy dwa razy z rzędu daje ten sam efekt (drugie wykonanie nic nie zmienia), podczas gdy powtórzenie `kubectl create` na już istniejącym obiekcie kończy się błędem `AlreadyExists`.

Generowanie szkieletu manifestu zamiast pisania od zera

Praktyczny kompromis między obydwoma trybami: używamy polecenia imperatywnego z flagą `--dry-run=client -o yaml`, żeby wygenerować poprawny składniowo szkielet manifestu bez wysyłania go do klastra (`client` oznacza, że walidacja i generowanie odbywają się wyłącznie lokalnie w `kubectl`, bez kontaktu z `kube-apiserver` — odmiennie od `--dry-run=server`, który wysyła żądanie do serwera i przechodzi pełną walidację łącznie z kontrolerami przyjęcia, ale bez trwałego zapisu). Przekierowujemy wynik do pliku, dopisujemy ręcznie to, czego generator nie umie wygenerować (limity zasobów, sondy, zmienne środowiskowe), i dopiero ten plik trafia do `kubectl apply` oraz do repozytorium.

```
kubectl create deployment katalog-produktow --image=git.zielona.example/zielona/katalog-produktow:2.4.1 --dry-run=client -o yaml > wdrozenie.yaml
```

Częste błędy

Wykonywanie `kubectl delete` bez sprawdzenia bieżącego kontekstu. Polecenie usuwające coś na klastrze testowym w zamierzeniu, wykonane przypadkiem na kontekście `produkcja` (bo ktoś zapomniał przełączyć), nie pyta drugi raz o potwierdzenie.

Sprawdzanie logów po restarcie kontenera bez `--previous`. Świeżo zrestartowany kontener ma czyste logi od nowego startu — przyczyna poprzedniej awarii jest widoczna wyłącznie w logach poprzedniej instancji.

Edytowanie obiektu bezpośrednio poleceniem `kubectl edit` na produkcji. Zmiana wchodzi w życie natychmiast, ale znika przy najbliższym `kubectl apply -f` z manifestu w repozytorium, który jej nie zawiera — źródło niespójności trudnej do zdiagnozowania później.

Poleganie na `kubectl run` do tworzenia czegokolwiek trwałego. Tworzy gołego poda, bez ReplicaSetu i bez mechanizmu samoleczenia po awarii — jeśli węzeł padnie, ten pod nie wstanie nigdzie indziej.

□□ Ćwiczenia

> Zadania sprawdzające.

1. Wypisujemy wszystkie konteksty zapisane w naszym `~/.kube/config` i wskazujemy, który jest aktualnie aktywny.
2. Generujemy poleceniem `kubectl create deployment ... --dry-run=client -o yaml` szkielet Deploymentu dla obrazu `nginx:1.27` i zapisujemy go do pliku, nie wysyłając niczego do klastra.
3. Wyjaśniamy różnicę między `kubectl logs` a `kubectl logs --previous` i podajemy sytuację, w której to drugie jest jedynym sposobem zdiagnozowania awarii.
4. Opisujemy, co dokładnie robi `kubectl apply -f` inaczej niż `kubectl replace -f`, i dlaczego to pierwsze jest bezpieczniejsze przy współpracy wielu osób nad tym samym klastrem.

Powiązane: Po co Kubernetes, skoro mamy Dockera · Obiekty, manifesty i przestrzenie nazw · Deployment, ReplicaSet i aktualizacje bez przerwy · Codzienne komendy

Źródła: dokumentacja Kubernetes, „kubectl Reference Docs" (kubernetes.io/docs/reference/generated/kubectl/kubectl-commands/) · „kubectl Cheat Sheet" (kubernetes.io/docs/reference/kubectl/cheatsheet/) · „Object Management Using kubectl" (kubernetes.io/docs/concepts/overview/working-with-objects/object-management/)

Revision #34
Created 2026-09-25 10:32:55 UTC by Tomasz Konieczny
Updated 2026-10-26 14:16:56 UTC by Tomasz Konieczny