Wolumeny, PV, PVC i StorageClass

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

Kubernetes rozdziela wolumen fizycznie istniejący w infrastrukturze od jego żądania przez aplikację — ta strona pokazuje, po co to rozdzielenie istnieje i jak z niego korzystamy krok po kroku, od `emptyDir` *do dynamicznego udostępniania przez StorageClass.*

Skróty użyte na tej stronie (8)

Skrót	Co znaczy
CNI	Container Network Interface: wtyczka, która daje sieć podom w Kubernetesie
CSI	Container Storage Interface: wtyczka, która daje dyski podom w Kubernetesie
HDD	Hard Disk Drive: tradycyjny dysk twardy z wirującymi talerzami i głowicą, wolniejszy od SSD, ale tańszy za gigabajt, typowo używany jako warstwa pojemnościowa
NFS	Network File System: udostępnianie katalogów po sieci w świecie Linuksa
PV	Physical Volume: dysk lub partycja oddana pod LVM; w Kubernetesie PersistentVolume, trwały wolumen
PVC	PersistentVolumeClaim: zamówienie trwałego dysku przez aplikację w Kubernetesie
REST	sposób budowania API przez zwykłe żądania HTTP (GET, POST) zamiast osobnego protokołu
SSD	Solid State Drive: dysk półprzewodnikowy

Fiszka

PLIK KONFIGURACYJNY

Utworzenie poda z emptyDir

```
apiVersion: v1
kind: Pod
```

```
metadata:
  name: NazwaPoda
spec:
  containers:
    - name: NazwaKontenera
      image: NazwaObrazu
      volumeMounts:
        - name: cache
          mountPath: /cache
  volumes:
    - name: cache
      emptyDir: {}
```

Tworzymy wolumen istniejący tylko przez cały czas życia poda, dzielony między kontenerami tego samego poda — znika bezpowrotnie razem z podem.

□□ TYLKO ODCZYT

Wylistowanie PersistentVolume w klastrze: kubectl get pv

```
kubectl get pv
```

Sprawdzamy, jakie woluminy fizyczne są zarejestrowane w klastrze, jaki mają rozmiar, tryb dostępu i status — kolumna `STATUS` mówi, czy wolumen jest wolny, czy już przypisany do jakiegoś żądania.

Przykład:

```
NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM                     STORAGECLASS   AGE
pv-baza-danych-01   20Gi       RWO            Retain           Bound    produkcja/baza-
danych-pvc   nfs-baza      90d
```

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

□□ PLIK KONFIGURACYJNY

Utworzenie PersistentVolumeClaim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: NazwaUslugi-pvc
  namespace: NazwaPrzestrzeni
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: NazwaKlasy
  resources:
    requests:
      storage: 10Gi
```

Zgłaszamy żądanie przestrzeni dyskowej o określonym rozmiarze i trybie dostępu, nie dotykając w ogóle tego, gdzie fizycznie ten dysk leży — o to dbają PersistentVolume i StorageClass opisane niżej.

□□ PLIK KONFIGURACYJNY

Podpięcie PVC do poda

```
apiVersion: v1
kind: Pod
metadata:
  name: NazwaPoda
  namespace: NazwaPrzestrzeni
spec:
  containers:
    - name: NazwaKontenera
      image: NazwaObrazu
      volumeMounts:
        - name: dane
          mountPath: /var/lib/dane
  volumes:
    - name: dane
      persistentVolumeClaim:
        claimName: NazwaUslugi-pvc
```

Montujemy przestrzeń z PVC pod konkretną ścieżką wewnątrz kontenera — z punktu widzenia aplikacji wygląda to jak zwykły katalog na dysku.

PVC = PersistentVolumeClaim: zamówienie trwałego dysku przez aplikację w Kubernetesie

TYLKO ODCZYT

Wylistowanie dostępnych klas magazynowania: kubectl get storageclass

```
kubectl get storageclass
```

Sprawdzamy, jakie klasy magazynowania są zdefiniowane w klastrze i która z nich jest domyślna — domyślną poznajemy po adnotacji widocznej w nazwie z gwiazdką w tym poleceniu.

Przykład:

```
NAME                 PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE   AGE
standard (default)   ebs.csi.aws.com         Delete          WaitForFirstConsumer   120d
nfs-baza             nfs.csi.k8s.io          Retain          Immediate              120d
```

kubectl = Kubernetes control: główne narzędzie Kubernetesa

ZMIENIA SYSTEM

Rozszerzenie wolumenu: kubectl edit / kubectl patch

```
kubectl patch pvc NazwaUslugi-pvc -n NazwaPrzestrzeni --type merge -p
'{"spec":{"resources":{"requests":{"storage":"20Gi"}}}}'
```

Zwiększamy zadeklarowany rozmiar PVC — czy operacja faktycznie powiększy dysk bez przestoju, zależy od tego, czy wybrana StorageClass wspiera rozszerzanie, opisane niżej.

PVC = PersistentVolumeClaim: zamówienie trwałego dysku przez aplikację w Kubernetesie · **kubectl** = Kubernetes control: główne narzędzie Kubernetesa

TYLKO ODCZYT

Sprawdzenie statusu PVC po rozszerzeniu

```
kubectl get pvc NazwaUslugi-pvc -n NazwaPrzestrzeni -o jsonpath='{.status.conditions}'
```

Sprawdzamy, czy operacja rozszerzenia zakończyła się (`FileSystemResizePending` oznacza, że dysk urósł, ale system plików wewnątrz poda jeszcze czeka na zmianę rozmiaru).

PVC = PersistentVolumeClaim: zamówienie trwałego dysku przez aplikację w Kubernetesie · **kubectl** = Kubernetes control: główne narzędzie Kubernetesa

Z życia wzięte

Z ŻYCIA WZIĘTE

O 12:15 druga replika bazy kolejki nie chce wystartować na klastrze produkcyjnym i wisi w fazie `Pending` mimo wolnych zasobów na wszystkich węzłach. Tomek sprawdza zdarzenia poda i widzi komunikat o niemożności zamontowania wolumenu, bo PVC ma tryb dostępu `ReadWriteOnce`, a druga replika trafiła planistą na inny węzeł niż pierwsza. Po zmianie magazynu na obsługujący `ReadWriteMany` obie repliki montują ten sam wolumen jednocześnie, bez żadnego dalszego błędu.

Czym jest wolumen w Kubernetesie

Pojęcie: wolumen (volume). Katalog z danymi dostępny wewnątrz jednego lub kilku kontenerów danego poda, o czasie życia i pochodzeniu zależnym od tego, jakiego typu wolumenu użyjemy. To jest inne pojęcie niż wolumen Dockera: wolumen Dockera żyje niezależnie od kontenera na tym samym hoście i pozostaje po jego usunięciu, dopóki ktoś go jawnie nie usunie. Wolumen Kubernetesa w wersji podstawowej (`emptyDir`) żyje dokładnie tak długo jak pod, do którego jest przypięty, i znika razem z nim — trwałość danych ponad cykl życia jednego poda wymaga świadomego sięgnięcia po PersistentVolume, opisany niżej, a nie jest domyślną cechą samego pojęcia wolumenu.

emptyDir

Pojęcie: emptyDir. Pusty katalog tworzony w momencie startu poda i usuwany bezpowrotnie w momencie jego usunięcia — restart kontenera wewnątrz tego samego poda nie kasuje `emptyDir`, ale usunięcie całego poda tak. Typowe zastosowanie to przestrzeń robocza dzielona między kilkoma kontenerami tego samego poda (np. kontener główny i sidecar czytający jego logi) albo bufor na dane tymczasowe, których utrata nie ma żadnych konsekwencji.

hostPath

Pojęcie: hostPath. Wolumen wskazujący bezpośrednio na katalog istniejący na dysku węzła, na którym akurat działa pod. Dane w `hostPath` przeżywają usunięcie poda, ale tylko dopóty, dopóki kolejny pod tej samej aplikacji trafi na ten sam węzeł — scheduler nie ma żadnego obowiązku umieszczać kolejnych replik zawsze na tym samym węźle, więc pod zrestartowany na innym węźle widzi pusty katalog, mimo że dane fizycznie leżą gdzie indziej. `hostPath` na produkcji jest zwykle błędem z dwóch powodów naraz: łamie założenie, że pod może wylądować na dowolnym węźle bez utraty danych, i otwiera furtkę bezpieczeństwa — kontener z dostępem do `hostPath` wskazującego na wrażliwy katalog systemowy węzła (np. `/etc` albo gniazdo dockerowe) potrafi wpłynąć na cały węzeł, nie tylko na siebie.

PersistentVolume i PersistentVolumeClaim

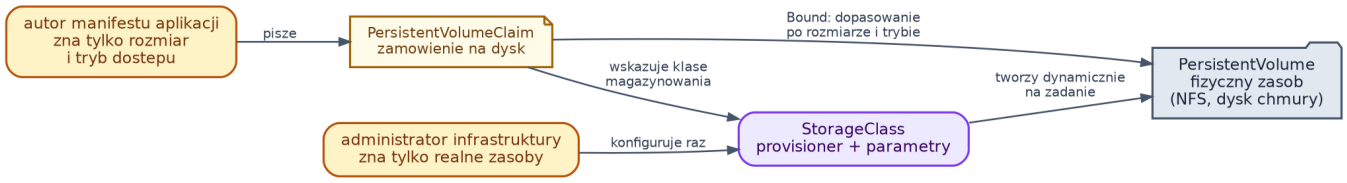
To jest sedno tej strony: Kubernetes celowo rozdziela dwie role, które w prostszych systemach są jedną rzeczą.

Pojęcie: PersistentVolume (PV). Obiekt reprezentujący fizycznie istniejącą przestrzeń dyskową — dysk w chmurze, zasób NFS, wolumen CSI — zarejestrowaną w klastrze jako zasób dostępny do użycia. PV tworzy zwykle administrator klastra (albo, w wariancie dynamicznym opisanym niżej, tworzy go automatycznie kontroler StorageClass) i istnieje niezależnie od jakiegokolwiek konkretnego poda czy aplikacji.

Pojęcie: PersistentVolumeClaim (PVC). Obiekt reprezentujący żądanie przestrzeni dyskowej zgłoszone przez aplikację — ile miejsca potrzebuje, w jakim trybie dostępu,

> ewentualnie z jakiej klasy magazynowania. PVC nie jest samym dyskiem, tylko zamówieniem na dysk.

Rozdzielenie tych dwóch ról ma konkretny cel organizacyjny: osoba pisząca manifest aplikacji (PVC) nie musi wiedzieć nic o infrastrukturze pod spodem — nie zna adresu serwera NFS, nie zna identyfikatora woluminu w chmurze, nie wie, czy dysk jest SSD czy HDD. Zna tylko to, ile miejsca potrzebuje i jak bardzo potrzebuje go współdzielić. Administrator infrastruktury po drugiej stronie zarządza realnymi zasobami (PV) i nie musi wiedzieć nic o konkretnych aplikacjach, które z nich skorzystają. Kubernetes w środku dopasowuje jedno do drugiego (`Bound`) na podstawie zgodności rozmiaru, trybu dostępu i klasy magazynowania — dokładnie tak jak zamówienie w sklepie (PVC) dopasowuje się do towaru na półce (PV) bez potrzeby, żeby klient wiedział, z której hurtowni towar przyjechał.



Autor manifestu pisze tylko PVC, administrator konfiguruje tylko StorageClass, a Kubernetes dopasowuje PVC do PV automatycznie (kliknięcie otwiera pełny rozmiar)

StorageClass i dynamiczne udostępnianie

> **Pojęcie: StorageClass.** Obiekt opisujący jeden rodzaj przestrzeni dyskowej dostępny w danej infrastrukturze wraz z parametrami jej tworzenia — i, co ważniejsze, wskazujący sterownik (provisioner), który potrafi taki dysk utworzyć na żądanie, automatycznie, bez ręcznej pracy administratora.

Bez StorageClass administrator musiałby ręcznie tworzyć obiekt PV za każdym razem, gdy aplikacja potrzebuje nowego kawałka przestrzeni dyskowej — podejście, które działa przy kilku aplikacjach, i przestaje skalować się przy setkach. Z dynamicznym udostępnianiem PVC odwołujący się do danej StorageClass sam wyzwala utworzenie odpowiadającego mu PV w tle, w momencie zgłoszenia żądania — administrator konfiguruje StorageClass raz, a każdy kolejny PVC obsługuje się sam.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: NazwaKlasy
provisioner: nfs.csi.k8s.io
reclaimPolicy: Retain
allowVolumeExpansion: true
parameters:
  server: NazwaSerweraNFS
  share: /export/dane
```

> (gdzie `NazwaSerweraNFS` to serwer udostępniający zasób NFS dla klastra; nie wskazujemy tu `bkp01`, bo to serwer kopii, a nie magazyn danych aplikacji: kopie z klastrów trafiają na niego przez restic i rest-server, z kontem REST nazwanym jak repozytorium, według strony Serwery kopii bkp01 i bkp02: restic, kopia krzyżowa i niezmienność)

Tryby dostępu

> **Pojęcie: tryb dostępu (access mode).** Deklaracja tego, ilu węzłom naraz i w jakim trybie wolno zamontować dany wolumen — nie jest to uprawnienie systemu plików w rozumieniu `chmod`, tylko ograniczenie na poziomie samej infrastruktury magazynowej.

Tryb	Co oznacza	Konsekwencja dla skalowania
`ReadWriteOnce` (RWO)	odczyt i zapis z jednego węzła naraz	wiele podów na tym samym węźle mogą go współdzielić, ale przeniesienie na inny węzeł wymaga odmontowania z poprzedniego — częsta przyczyna, dla której pod utyka w `Pending` przy próbie przeniesienia
`ReadOnlyMany` (ROX)	odczyt z wielu węzłów naraz, bez zapisu	dobre do danych współdzielonych i niezmiennych, np. statycznych zasobów serwowanych przez wiele replik
`ReadWriteMany` (RWX)	odczyt i zapis z wielu węzłów naraz	wymaga systemu plików faktycznie wspierającego dostęp współbieżny (NFS, CephFS) — większość dysków blokowych w chmurze tego nie oferuje

Tryb	Co oznacza	Konsekwencja dla skalowania
`ReadWriteOncePod` (RWOP)	odczyt i zapis, ale z dokładnie jednego poda w całym klastrze	ściślejsza wersja RWO, wprowadzona żeby wykluczyć nawet przypadkowe współdzielenie między dwoma podami na tym samym węźle — istotne dla aplikacji, które same zakładają wyłączność na zapis (część silników baz danych)

Uważaj: aplikacja skalowana do kilku replik z wolumenem w trybie `ReadWriteOnce` nie skaluje się poziomo tak, jak można by się spodziewać — druga i kolejne repliki na innym węźle utykają w `Pending`, dopóki nie trafią na ten sam węzeł co pierwsza, albo dopóki aplikacja nie przejdzie na magazyn wspierający `ReadWriteMany`.

Polityka odzyskiwania

> **Pojęcie: polityka odzyskiwania (reclaim policy).** Decyzja o tym, co dzieje się z PersistentVolume w momencie, gdy powiązany z nim PVC zostaje usunięty. `Delete` usuwa też fizyczny wolumen w infrastrukturze pod spodem — wygodne dla danych efemerycznych, niebezpieczne dla danych, których utrata boli. `Retain` zostawia wolumen i jego dane nietknięte, oznaczając go jako `Released` — wymaga ręcznej decyzji administratora, co dalej (ponowne podłączenie do nowego PVC albo ręczne usunięcie), ale nigdy nie kasuje danych automatycznie. Domyślna polityka zależy od tego, czy PV powstał ręcznie (`Retain`) czy dynamicznie przez StorageClass (najczęściej `Delete`, chyba że StorageClass jawnie ustawia inaczej) — warto to sprawdzić świadomie dla każdej klasy magazynowania używanej do danych, których nie da się łatwo odtworzyć.

Rozszerzenie wolumenu

Rozszerzenie PVC, pokazane w Fiszce, działa tylko wtedy, gdy StorageClass ma ustawione `allowVolumeExpansion: true` — bez tego pola żądanie zwiększenia rozmiaru PVC zostaje odrzucone od razu. Po udanym rozszerzeniu na poziomie infrastruktury dysk często wymaga jeszcze rozszerzenia systemu plików wewnątrz niego, co przy nowoczesnych sterownikach CSI dzieje się automatycznie przy najbliższym zamontowaniu (status `FileSystemResizePending` sygnalizuje właśnie ten pośredni stan) — samo powiększenie dysku po stronie infrastruktury nie oznacza jeszcze, że aplikacja widzi więcej wolnego miejsca.

CSI w skrócie

> **Pojęcie: CSI (Container Storage Interface).** Standardowy interfejs, przez który Kubernetes komunikuje się z dowolnym systemem magazynowania — dyskiem w chmurze, macierzą on-premise, systemem rozproszonym — bez potrzeby wbudowywania kodu obsługi każdego z nich bezpośrednio w Kubernetesa. Sterownik CSI to zewnętrzny komponent, instalowany osobno dla każdego rodzaju magazynu, który tłumaczy uniwersalne żądania Kubernetesa (utwórz wolumen, zamontuj, rozszerz) na konkretne operacje danej infrastruktury. To dokładnie ten sam mechanizm rozszerzalności co CNI dla sieci, opisany na stronie Sieć klastra i polityki sieciowe, tylko dla magazynu danych.

Częste błędy

Oczekiwanie, że hostPath przetrwa przeniesienie poda na inny węzeł. Dane fizycznie zostają na starym węźle — nowy pod na innym węźle widzi pusty katalog, mimo że nic nie zgłasza błędu.

PVC w trybie ReadWriteOnce podpięty do aplikacji skalowanej poziomo na wielu węzłach. Pierwsza replika startuje poprawnie, każda kolejna na innym węźle utyka w `Pending` z komunikatem o niemożności zamontowania wolumenu, opisanym na stronie Diagnostyka: objaw, przyczyna, pierwsze kroki.

Reclaim policy Delete na danych, których nie da się odtworzyć. Usunięcie PVC w ramach porządkowania środowiska kasuje bezpowrotnie także fizyczny dysk — warto świadomie ustawiać `Retain` dla każdej klasy magazynowania obsługującej dane produkcyjne, zamiast polegać na domyślnym zachowaniu sterownika.

□□ Ćwiczenia

> Ćwiczenia zakładają klaster testowy z zainstalowanym sterownikiem CSI i co najmniej jedną StorageClass wspierającą dynamiczne udostępnianie.

Ćwiczenie 1 — cykl życia PVC i PV

```
kubectl apply -f pvc-test.yaml
kubectl get pvc,pv
kubectl delete pvc test-pvc
kubectl get pv
```

Tworzymy PVC, obserwujemy automatycznie utworzony PV i sprawdzamy, co się z nim dzieje po usunięciu PVC, w zależności od ustawionej polityki odzyskiwania.

Odpowiedź

Przy `reclaimPolicy: Delete` PV znika razem z PVC, a fizyczny dysk w infrastrukturze zostaje skasowany. Przy `reclaimPolicy: Retain` PV zostaje w stanie `Released`, dane pozostają nietknięte, ale nowy PVC nie podłączy się do niego automatycznie bez ręcznej interwencji.

Pytania kontrolne

1. Dlaczego administrator infrastruktury i autor manifestu aplikacji nie muszą znać nawzajem swoich szczegółów, mimo że oboje zajmują się tym samym dyskiem?

Odpowiedź

Bo PV i PVC to dwie osobne warstwy abstrakcji — PVC opisuje wyłącznie potrzebę (rozmiar, tryb dostępu), a PV opisuje wyłącznie ofertę (konkretny zasób infrastruktury). Kubernetes dopasowuje jedno do drugiego, więc żadna ze stron nie musi znać szczegółów drugiej.

2. Dlaczego aplikacja z trzema replikami i wolumenem ReadWriteOnce nie skaluje się tak, jak oczekiwano?

Odpowiedź

ReadWriteOnce pozwala na zapis z jednego węzła naraz — druga i trzecia replika na innych węzłach nie mogą zamontować tego samego wolumenu i utykają w stanie `Pending`, dopóki nie trafią na ten sam węzeł co pierwsza replika albo dopóki magazyn nie zostanie zmieniony na wspierający ReadWriteMany.

Powiązane: Sieć klastra i polityki sieciowe · Diagnostyka: objaw, przyczyna, pierwsze kroki

Źródła: dokumentacja kubernetes.io, rozdziały „Volumes", „Persistent Volumes", „Storage Classes" i „Volume Snapshots" · dokumentacja Container Storage Interface (kubernetes-csi.github.io)

Revision #33
Created 2026-09-25 10:33:08 UTC by Tomasz Konieczny
Updated 2026-10-26 14:17:10 UTC by Tomasz Konieczny