Pamięć masowa w Kubernetesie

inż. Wojciech Baranowski, 184574 inż. Michał Łubiński, 184603

6 czerwca 2024

Spis treści

1 Ephemeral storage **3**
1.1 Definicja . . . 3
1.1.1 Przykładowe zastosowania . . . 3

2 Persistent storage **3**
2.1 Definicja . . . 3
2.1.1 Przykładowe zastosowania . . . 4
2.2 PersistentVolume (PV) . . . 4
2.2.1 Definicja . . . 4
2.2.2 Przykład . . . 4
2.3 PersistentVolumeClaim (PVC) . . . 5
2.3.1 Definicja . . . 5
2.3.2 Przykład . . . 5

3 Storage class **6**
3.1 Definicja . . . 6
3.2 Przykład . . . 6
3.3 Storage blokowy . . . 7
3.3.1 Definicja . . . 7
3.3.2 Zastosowanie . . . 7
3.3.3 Zalety . . . 7
3.3.4 Wady . . . 8
3.4 Storage plikowy . . . 8
3.4.1 Definicja . . . 8
3.4.2 Zastosowanie . . . 8
3.4.3 Zalety . . . 8
3.4.4 Wady . . . 8

4 Mechanizmy provisioningowe **9**
4.1 Longhorn . . . 9
4.1.1 Charakterystyka . . . 9
4.1.2 Zalety . . . 9
4.1.3 Wady . . . 9

4.2 GlusterFS . . . 9
4.2.1 Charakterystyka . . . 9
4.2.2 Zalety . . . 10
4.2.3 Wady . . . 10
4.3 Ceph RBD (RADOS Block Device) . . . 10
4.3.1 Charakterystyka . . . 10
4.3.2 Zalety . . . 10
4.3.3 Wady . . . 10
4.4 Ceph FS (Ceph Filesystem) . . . 11
4.4.1 Charakterystyka . . . 11
4.4.2 Zalety . . . 11
4.4.3 Wady . . . 11
4.5 iSCSI . . . 11
4.5.1 Charakterystyka . . . 11
4.5.2 Zalety . . . 11
4.5.3 Wady . . . 12
4.6 Porównanie . . . 12

5 Źródła **12**

1 Ephemeral storage

1.1 Definicja

Pamięć efemeryczna jest tymczasowa i związana bezpośrednio z instancją maszyny wirtualnej (VM). Oto kluczowe cechy i zastosowania pamięci efemerycznej:

- **Tymczasowość**: Dane przechowywane w pamięci efemerycznej są usuwane po zatrzymaniu, ponownym uruchomieniu lub usunięciu instancji VM. Przykładem może być lokalny dysk twardy maszyny wirtualnej.
- **Wysoka wydajność**: Często używana do przechowywania danych, które wymagają szybkiego dostępu.
- **Koszt**: Zwykle tańsza w porównaniu do pamięci trwałej, co czyni ją ekonomicznym wyborem dla danych krótkoterminowych.
- **Brak trwałości**: Nie jest przeznaczona do przechowywania danych, które muszą przetrwać restart lub usunięcie instancji.

1.1.1 Przykładowe zastosowania

- Tymczasowe przechowywanie danych sesji w aplikacjach internetowych.
- Pliki tymczasowe generowane podczas obliczeń i analizy danych.
- Cache aplikacji, które mogą być łatwo odtworzone.

2 Persistent storage

2.1 Definicja

Pamięć trwała jest zaprojektowana do długoterminowego przechowywania danych, niezależnie od stanu instancji VM. Oto jej kluczowe cechy i zastosowania:

- **Trwałość**: Dane przechowywane w pamięci trwałej przetrwają restart, zatrzymanie i usunięcie instancji VM.
- **Bezpieczeństwo**: Często oferuje funkcje takie jak szyfrowanie, kopie zapasowe i replikacja w celu zapewnienia bezpieczeństwa i dostępności danych.
- **Elastyczność**: Może być podłączona do różnych instancji VM, co pozwala na łatwe przenoszenie danych między instancjami.
- **Koszt**: Zwykle droższa w porównaniu do pamięci efemerycznej, ale zapewnia większą niezawodność i bezpieczeństwo dla danych krytycznych.
- **Przeznaczenie**: Idealna do przechowywania baz danych, aplikacji o znaczeniu krytycznym, danych użytkowników, plików konfiguracyjnych i innych trwałych danych.

2.1.1 Przykładowe zastosowania

- Bazy danych.
- Systemy plików przechowujące dane użytkowników (np. zdjęcia, dokumenty).
- Kopie zapasowe i archiwizacja danych.

2.2 PersistentVolume (PV)

2.2.1 Definicja

PersistentVolume (PV) to zasób w klastrze Kubernetes, który reprezentuje rzeczywiste wolumeny dyskowe. PV jest niezależny od węzłów klastra, co oznacza, że może być używany przez dowolną instancję aplikacji w klastrze. Kluczowe cechy PV to:

- **Trwałość**: PV istnieje niezależnie od cykli życia aplikacji, co oznacza, że wolumeny są dostępne nawet po ponownym uruchomieniu aplikacji lub migracji między węzłami klastra.
- **Dostępność**: Administratorzy mogą dynamicznie przydzielać i konfigurować PV w klastrze, co umożliwia zarządzanie zasobami przechowywania danych na poziomie klastra.
- **Różnorodność zasobów**: PV może reprezentować różne rodzaje przechowywania danych, takie jak dyski blokowe, NFS, czy chmurowe dyski.
- **Poziomy dostępu**: PV może być skonfigurowany z różnymi poziomami dostępu, takimi jak `ReadWriteOnce` (RWO) dla pojedynczego węzła, `ReadOnlyMany` (ROX) dla wielu węzłów do odczytu, czy `ReadWriteMany` (RWX) dla wielu węzłów do odczytu i zapisu.

2.2.2 Przykład

Poniżej znajduje się przykład definicji persistent volume'a w pliku YAML:

```
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "test-pv-1"
  namespace: "zsch"
  labels:
    type: "local"
spec:
  storageClassName: "manual"
  capacity:
    storage: "1Gi"
  accessModes:
    - "ReadWriteOnce"
  hostPath:
    path: "/mnt/data"
```

W powyższym przykładzie:

- **metadata**: Zawiera metadane dla PersistentVolume, takie jak jego nazwa, przestrzeń nazw, czy labelki.
- **storageClassName**: Określa nazwę klasy pamięci, z której ten wolumen ma korzystać (w tym przypadku pamięć przydzielana jest manualnie poprzez zadeklarowanie ścieżki w systemie plików hosta).
- **capacity**: Określa pojemność wolumenu (w tym przykładzie 1 gigabajt).
- **accessModes**: Określa tryby dostępu do wolumenu (w tym przykładzie ReadWriteOnce, czyli możliwy do zapisu i odczytu przez jedną instancję na raz).
- **hostPath**: określa ścieżkę w systemie plików hosta, w której ma zostać zamontowany zasób.

2.3 PersistentVolumeClaim (PVC)

2.3.1 Definicja

PersistentVolumeClaim (PVC) jest żądaniem zasobu PV przez aplikację w klastrze Kubernetes. PVC definiuje wymagania dotyczące przechowywania danych dla aplikacji, takie jak rozmiar wolumenu i żądany poziom dostępu. Kluczowe cechy PVC to:

- **Abstrakcja od zasobów fizycznych**: Aplikacje korzystają z PVC, aby określić swoje wymagania przechowywania danych, niezależnie od konkretnego zasobu PV.
- **Dynamiczne przydzielanie**: PVC może być dynamicznie powiązany z dostępnymi PV, które spełniają określone kryteria, dzięki czemu aplikacje nie muszą z góry konfigurować zasobów przechowywania danych.
- **Elastyczność**: Aplikacje mogą używać wielu PVC w klastrze, a PVC może być modyfikowane w trakcie działania aplikacji, co umożliwia dostosowywanie zasobów przechowywania danych do zmieniających się potrzeb aplikacji.

2.3.2 Przykład

Poniżej znajduje się przykład definicji persistent volume claim'a w pliku YAML:

```
apiVersion: "v1"
kind: "PersistentVolumeClaim"
metadata:
  name: "test-pvc"
  namespace: "zsch"
spec:
  storageClassName: "manual"
  accessModes:
    - "ReadWriteOnce"
  resources:
    requests:
      storage: "2Gi"
```

W powyższym przykładzie:

- **metadata**: Zawiera metadane dla PersistentVolumeClaim, takie jak jego nazwa czy przestrzeń nazw.
- **storageClassName**: Określa nazwę klasy pamięci, z której PVC ma korzystać (w tym przypadku pamięć jest zarządzana manualnie).
- **accessModes**: Określa tryb dostępu do wolumenu (w tym przykładzie ReadWriteOnce, czyli możliwy do zapisu i odczytu przez jedną instancję na raz).
- **resources**: Definiuje zasoby żądane przez PVC, w tym rozmiar wolumenu (w tym przykładzie 2 gigabajty).

3 Storage class

3.1 Definicja

W kubernetesie storage class jest zasobem używanym do dynamicznego provisioningu storage'u. Definiuje on sposób tworzenia persistent volumes za pomocą specyfikacji provisionera, klasy wydajności, parametrów oraz polityk odzysku (reclaim policies). Storage class umożliwia administratorom zdefiniowanie różnych klas pamięci masowej, które mogą być wykorzystywane przez użytkowników w zależności od ich potrzeb. Pozwala to na automatyczne zarządzanie i provisionowanie zasobów storage'u w klastrze.
Każda storage class posiada następujące atrybuty:

- **Provisioner:** Mechanizm, który tworzy volumy. Najpopularniejsze provisionery opisane zostały w rozdziale 4.
- **Parameters:** Parametry specyficzne dla danego provisionera, które określają szczegóły wolumenów, takie jak typ dysku, rozmiar, poziom redundancji, itp.
- **Reclaim Policy:** Polityka odzysku wolumenu, która określa, co zrobić z wolumenem po usunięciu podłączonego do niego persistent volume claim'u . Może to być `Retain` (zachowanie volumu), `Delete` (usunięcie volumu) lub `Recycle` (przywrócenie volumu do stanu pierwotnego).
- **Binding Mode:** Określa, kiedy wolumen jest provisionowany. `Immediate` powoduje natychmiastowe provisionowanie volumu, natomiast `WaitForFirstConsumer` odracza provisionowanie do momentu, kiedy pierwszy pod zażąda zasobu.

3.2 Przykład

Poniżej znajduje się przykład definicji storage class w pliku YAML:

```
kind: "StorageClass"
apiVersion: "storage.k8s.io/v1"
metadata:
  name: "longhorn"
```

```
provisioner: "driver.longhorn.io"
allowVolumeExpansion: true
reclaimPolicy: "Delete"
volumeBindingMode: "Immediate"
parameters:
  numberOfReplicas: "2"
  staleReplicaTimeout: "3600"
  fromBackup: ""
  fsType: "ext4"
```

W powyższym przykładzie:

- **name** - nazwa klasy storage'u (w tym przypadku `longhorn`).
- **provisioner:** - provisioner odpowiedzialny za tworzenie wolumenu, w tym przypadku Longhorn (w tym przypadku `driver.longhorn.io`).
- **parameters:** - specyficzne parametry dla provisionera, takie jak rodzaj systemu plików (ext4) czy liczba replik (2).
- **reclaimPolicy:** - polityka odzysku, (w tym przypadku `delete` - polityka, która usuwa wolumen po usunięciu PVC).
- **volumeBindingMode:** - parametr decydujący o momencie utworzenia wolumenu (w tym przypadku `Immediate` - wolumen zostanie utworzony natychmiastowo).

3.3 Storage blokowy

3.3.1 Definicja

Storage blokowy odnosi się do przechowywania danych w postaci bloków. Każdy blok jest niezależnie adresowany i może być przechowywany w różnych miejscach.

3.3.2 Zastosowanie

- Zazwyczaj używany do przechowywania danych dla baz danych i systemów wymagających wysokiej wydajności.
- Idealny dla aplikacji o dużej intensywności I/O, takich jak systemy transakcyjne.
- Używany jako podstawowy storage dla maszyn wirtualnych (VM) w środowiskach chmurowych.

3.3.3 Zalety

- **Wysoka wydajność:** Storage blokowy oferuje bardzo niskie opóźnienia i wysoką przepustowość, co jest kluczowe dla aplikacji wymagających dużej wydajności.
- **Elastyczność:** Może być łatwo podzielony na partycje i formatowany w dowolny system plików, co daje dużą swobodę w zarządzaniu danymi.
- **Bezpieczeństwo:** Może być używany z różnymi technologiami szyfrowania, co zwiększa bezpieczeństwo danych.

3.3.4 Wady

- **Kompleksowość zarządzania:** Zarządzanie storage'em blokowym może być bardziej skomplikowane, wymaga zaawansowanych umiejętności administracyjnych.
- **Koszty:** Może być droższy w porównaniu do innych form storage'u, szczególnie przy wysokich wymaganiach wydajnościowych.

3.4 Storage plikowy

3.4.1 Definicja

Storage plikowy przechowuje dane w postaci plików i katalogów, które są organizowane w strukturze hierarchicznej.

3.4.2 Zastosowanie

- Używany do przechowywania plików w aplikacjach o mniej intensywnych wymaganiach I/O.
- Idealny do współdzielenia plików w sieciach, np. w systemach współdzielenia plików czy serwerach NAS (Network Attached Storage).
- Często stosowany w środowiskach gdzie potrzebna jest współpraca nad plikami, takich jak biura i zespoły projektowe.

3.4.3 Zalety

- **Łatwość użycia:** Intuicyjna struktura plików i katalogów jest łatwa do zrozumienia i zarządzania.
- **Współdzielenie danych:** Idealny do środowisk, gdzie pliki muszą być łatwo udostępniane i współdzielone między użytkownikami.
- **Koszty:** Może być tańszy w porównaniu do storage'u blokowego, szczególnie w aplikacjach o mniejszych wymaganiach wydajnościowych.

3.4.4 Wady

- **Wydajność:** Storage plikowy zazwyczaj oferuje niższą wydajność w porównaniu do storage'u blokowego, co może być problematyczne dla aplikacji wymagających wysokiej wydajności I/O.
- **Skalowalność:** Może być mniej skalowalny niż storage blokowy, zwłaszcza w bardzo dużych środowiskach.
- **Ograniczenia systemów plików:** Jest ograniczony przez specyfikacje systemów plików, co może wpływać na elastyczność i wydajność.

4 Mechanizmy provisioningowe

4.1 Longhorn

4.1.1 Charakterystyka

Longhorn jest rozproszonym systemem przechowywania danych zaprojektowanym specjalnie do działania z Kubernetes. Działa w całości w przestrzeni użytkownika, co oznacza, że nie wymaga modyfikacji jądra systemu operacyjnego. Longhorn oferuje pełną integrację z kubernetesem, co ułatwia zarządzanie wolumenami bezpośrednio z poziomu platformy kubernetes. System automatycznie replikuje dane, zapewniając ich wysoką dostępność. Zarządzanie Longhornem jest intuicyjne dzięki interfejsowi użytkownika (UI) oraz narzędziom wiersza poleceń (CLI). Dodatkowo, Longhorn obsługuje snapshoty i backupy, co umożliwia tworzenie kopii zapasowych oraz przywracanie danych po awarii. Skalowanie systemu jest proste i polega na dodawaniu nowych węzłów do klastra.

4.1.2 Zalety

- Pełna integracja z Kubernetes.
- Prosta konfiguracja i zarządzanie przez UI oraz CLI.
- Automatyczna replikacja danych.
- Obsługa snapshotów i backupów.
- Łatwe skalowanie przez dodawanie nowych węzłów.

4.1.3 Wady

- Może nie być tak wydajny jak natywne rozwiązania działające w przestrzeni jądra.
- Wymaga dodatkowych zasobów na replikację danych.

4.2 GlusterFS

4.2.1 Charakterystyka

GlusterFS to skalowalny, rozproszony system plików, który umożliwia tworzenie woluminów rozproszonych na wielu serwerach, działających jako jeden spójny system plików. GlusterFS jest idealny do środowisk wymagających wysokiej skalowalności i dostępności. Replikacja i dystrybucja danych są wbudowane w system, co zapewnia ich niezawodność. Zarządzanie GlusterFS może być skomplikowane, ponieważ wymaga manualnej konfiguracji i jest realizowane głównie przez wiersz poleceń (CLI). System obsługuje różne typy wolumenów, takie jak replicated, distributed i striped, co daje elastyczność w zarządzaniu danymi. Dodatkowo, GlusterFS oferuje funkcje snapshotów i samoutrzymywania, co zwiększa jego niezawodność.

4.2.2 Zalety

- Wysoka skalowalność.
- Automatyczna replikacja i dystrybucja danych.
- Obsługa różnych typów wolumenów (replicated, distributed, striped).
- Funkcje snapshotów i samoutrzymywania.

4.2.3 Wady

- Skomplikowana konfiguracja i zarządzanie.
- Wymaga manualnej konfiguracji.
- Potencjalne problemy z wydajnością w dużych klastrach.

4.3 Ceph RBD (RADOS Block Device)

4.3.1 Charakterystyka

Ceph RBD jest częścią systemu Ceph i oferuje blokowy dostęp do przechowywania danych. Ceph jest rozproszonym systemem przechowywania danych, który zapewnia wysoką wydajność, skalowalność i dostępność. Ceph RBD umożliwia tworzenie urządzeń blokowych dostępnych przez sieć, co jest idealne dla aplikacji wymagających dużej wydajności, takich jak bazy danych czy maszyny wirtualne. System automatycznie replikuje dane i zarządza uszkodzonymi dyskami, co zapewnia wysoką dostępność i niezawodność. Zarządzanie Ceph RBD jest realizowane przez kompleksową konfigurację i narzędzia wiersza poleceń (CLI). Dodatkowo, Ceph RBD obsługuje snapshoty i klonowanie wolumenów, co ułatwia zarządzanie kopiami zapasowymi i wdrażanie nowych aplikacji.

4.3.2 Zalety

- Wysoka wydajność i skalowalność.
- Automatyczna replikacja i zarządzanie uszkodzonymi dyskami.
- Obsługa snapshotów i klonowania wolumenów.
- Idealny dla aplikacji wymagających dużej wydajności (bazy danych, VM).

4.3.3 Wady

- Kompleksowa konfiguracja i zarządzanie.
- Wymaga zaawansowanej wiedzy technicznej.
- Może być zasobożerny.

4.4 Ceph FS (Ceph Filesystem)

4.4.1 Charakterystyka

Ceph FS to rozproszony system plików będący częścią ekosystemu Ceph. Umożliwia tworzenie rozproszonych systemów plików dostępnych przez sieć, co jest idealne dla aplikacji wymagających współdzielonego dostępu do plików. Ceph FS zapewnia wysoką skalowalność i elastyczność, umożliwiając łatwe dodawanie nowych zasobów do systemu. System automatycznie replikuje dane i zarządza ich dostępnością, co zwiększa niezawodność i dostępność danych. Zarządzanie Ceph FS wymaga zaawansowanej konfiguracji i jest realizowane głównie przez wiersz poleceń (CLI). Ceph FS obsługuje również snapshoty i klonowanie, co ułatwia zarządzanie kopiami zapasowymi i wdrażanie nowych aplikacji.

4.4.2 Zalety

- Wysoka skalowalność i elastyczność.
- Automatyczna replikacja i zarządzanie dostępnością danych.
- Obsługa snapshotów i klonowania.
- Idealny dla aplikacji wymagających współdzielonego dostępu do plików.

4.4.3 Wady

- Skomplikowana konfiguracja i zarządzanie.
- Wymaga zaawansowanej wiedzy technicznej.
- Może być zasobożerny.

4.5 iSCSI

4.5.1 Charakterystyka

iSCSI (Internet Small Computer System Interface) to protokół umożliwiający przesyłanie poleceń SCSI przez sieć IP. Dzięki temu możliwy jest zdalny dostęp do urządzeń blokowych, co jest powszechnie stosowane w środowiskach korporacyjnych. iSCSI charakteryzuje się wysoką wydajnością i niskimi opóźnieniami, szczególnie w dobrze zarządzanych sieciach. Protokół ten jest szeroko kompatybilny i wspierany przez wiele urządzeń i systemów operacyjnych. Zarządzanie iSCSI wymaga konfiguracji sieciowej i jest realizowane przez wiersz poleceń (CLI). Chociaż iSCSI nie oferuje natywnej replikacji, można ją uzyskać za pomocą dodatkowych narzędzi. iSCSI jest idealne dla aplikacji wymagających wysokiej wydajności i niskiego opóźnienia, takich jak bazy danych czy środowiska wirtualizacji.

4.5.2 Zalety

- Wysoka wydajność i niskie opóźnienia.
- Szeroka kompatybilność i wsparcie.
- Idealny dla aplikacji wymagających wysokiej wydajności (bazy danych, wirtualizacja).

4.5.3 Wady

- Wymaga konfiguracji sieciowej i zarządzania.
- Brak natywnej replikacji (wymaga dodatkowych narzędzi).
- Potencjalne problemy z dostępnością i niezawodnością w przypadku awarii sieci.

4.6 Porównanie

W celu porównania wydajności poszczególnych rozwiązań wykorzystany został klaster kubernetesowy złożony z trzech węzłów typu master i szesnastu węzłów typu worker. Na klastrze zostały uruchomione poszczególne rozwiązania (Longhorn, GlusterFS, Ceph RBD, CephFS). Do testów posłużyła baza danych SQLite, dla której podmontowano persistent volume claim'a o polityce dostępu wskazanej dla danego rodzaju storage'u (RWO dla blokowego i RWX dla plikowego). Na bazie dokonywano operacji I/O równolegle przez 1, 2, 4 i 8 wątków. Proces testowania orkiestrowany był przy pomocy narzędzia Phoronix Test Suite. Poniższa tabela przedstawia rezultaty:

Liczba wątków	Longhorn	GlusterFS	Ceph RBD	CephFS
1	238.461	380.979	173.766	186.274
2	433.586	377.084	365.782	207.837
4	616.192	422.790	477.536	229.304
8	905.618	737.335	603.924	286.458

Z pomocą powyższych wyników można wyciągnąć następujace wnioski:

- Storage blokowy wykorzystujący politykę RWO jest wydajniejszy w obsłudze pojedynczego wątku, niż storage plikowy.
- Storage plikowy wykorzystujący politykę RWX jest wydajniejszy w obsłudze wielu wątków, niż storage blokowy.
- Rozwiązania udostępnione przez system Ceph są istotnie wydajniejsze, niż pozostałe provisionery.

5 Źródła

- slajdy wykładowe
- materiały udostępnione przez Centrum Usług Informatycznych Politechniki Gdańskiej
- https://kubernetes.io/docs/home/
- https://longhorn.io
- https://www.gluster.org
- https://ceph.io
- https://en.wikipedia.org/wiki/ISCSI
- wiedza własna