10 Zasoby StatefulSet: wdrażanie replikowanych aplikacji stanowych

W tym rozdziale:

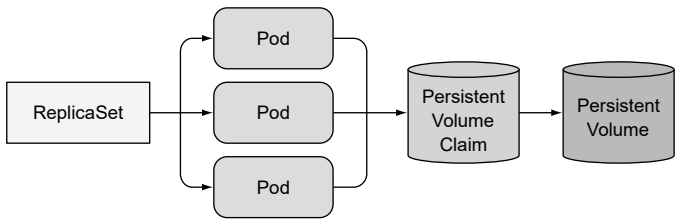
- Wdrażanie klastrowych aplikacji stanowych
- Udostępnianie oddzielnej pamięci masowej dla każdego wystąpienia replikowanego podu
- Gwarantowanie stabilnej nazwy i nazwy hosta dla replik podów
- Uruchamianie i zatrzymywanie replik podów w przewidywalnej kolejności
- Odkrywanie partnerów za pośrednictwem rekordów DNS SRV

Wiemy już, jak uruchamiać pody bezstanowe (*stateless*), zarówno w pojedynczym wystąpieniu, jak i replikowane, a nawet pody stanowe (*stateful*) wykorzystujące trwałą pamięć masową. Możemy uruchamiać wiele zreplikowanych wystąpień serwera WWW i pojedyncze wystąpienie bazy danych, które używa trwałej pamięci masowej, zapewnianej albo jako zwykłe woluminy podu, albo przez zasoby PersistentVolume powiązane

z obiektem PersistentVolumeClaim. Czy moglibyśmy zastosować ReplicaSet, aby uzyskać replikę podu bazy danych?

10.1 Replikowanie podów stanowych

Obiekty ReplicaSet tworzą wiele replik podów z pojedynczego szablonu. Repliki te nie różnią się między sobą, jeśli nie liczyć ich nazwy i adresu IP. Jeśli szablon podu zawiera wolumin odwołujący się do określonego obiektu PersistentVolumeClaim (PVC), wszystkie repliki w tym zasobie ReplicaSet będą używać dokładnie tego samego żądania PVC, a tym samym będą miały powiązany ten sam obiekt PersistentVolume przypisany przez żądanie (patrz rysunek 10.1).



Rysunek 10.1 Wszystkie pody z tego samego zasobu ReplicaSet zawsze używają tego samego PersistentVolumeClaim oraz PersistentVolume

Ponieważ odwołanie do żądania znajduje się w szablonie podu, który służy do wytłaczania wielu replik tego podu, nie możemy zapewnić, że każda replika będzie używać swojego własnego, oddzielnego PersistentVolumeClaim. Nie możemy użyć zasobu ReplicaSet do uruchamiania rozproszonego magazynu danych, w którym każde wystąpienie potrzebuje swojej własnej, oddzielnej pamięci masowej – a przynajmniej nie przy użyciu pojedynczego obiektu ReplicaSet. Uczciwie mówiąc, żaden spośród obiektów API, które pokazaliśmy do tej pory, nie umożliwia uruchomienia takiego magazynu danych. Potrzebujemy czegoś innego.

10.1.1 Uruchamianie wielu replik, każdej z oddzielną pamięcią masową

Jak moglibyśmy uruchomić wiele replik podu i zapewnić, że każdy będzie używać swojego własnego woluminu pamięci masowej? Obiekty ReplicaSet tworzą dokładne kopie (repliki) podu; tym samym nie możemy użyć ich dla podów takiego typu. Czego możemy użyć?

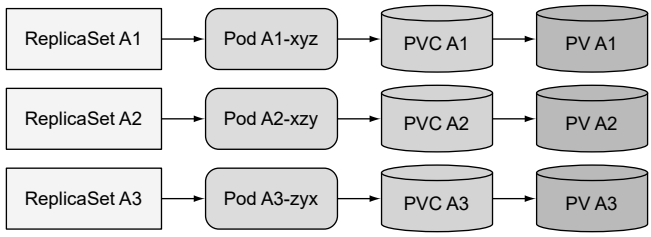
RĘCZNE TWORZENIE PODÓW

Moglibyśmy utworzyć pody ręcznie i w każdym z nich użyć jego własnego żądania PersistentVolumeClaim. Jednak ponieważ nie będzie o nie dbał żaden kontroler replikacji (ReplicaSet), musielibyśmy nimi zarządzać i odtwarzać je ręcznie, gdy znikną (jak w przypadku awarii węzła). Tym samym nie jest to realna opcja.

Użycie dedykowanego zasobu ReplicaSet dla każdego wystąpienia podu

Zamiast bezpośredniego tworzenia podów moglibyśmy utworzyć wiele obiektów ReplicaSet – po jednym dla każdego podu, ustawiając licznik pożądanych replik na jeden, przy czym szablon podu w każdym obiekcie ReplicaSet będzie się odwoływał do dedykowanego żądania PersistentVolumeClaim (patrz rysunek 10.2).

Wprawdzie to podejście zapewni automatyczne rozmieszczanie w przypadku awarii węzłów lub przypadkowego usunięcia podu, jednak jest znacznie bardziej pracochłonne, gdy porównamy je z użyciem pojedynczego obiektu ReplicaSet. Dla przykładu zastanówmy się, jak moglibyśmy skalować liczbę podów w tej sytuacji. Nie możemy zmienić pożądanej liczby replik – musimy za to utworzyć dodatkowe zasoby ReplicaSet.

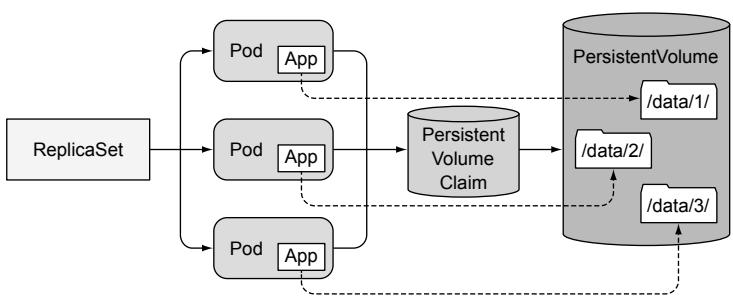


Rysunek 10.2 Używanie dedykowanego obiektu ReplicaSet dla każdego wystąpienia podu

Użycie wielu obiektów ReplicaSet nie jest zatem najlepszym rozwiązaniem. Może jednak moglibyśmy użyć pojedynczego obiektu ReplicaSet i zapewnić, aby każde wystąpienie podu utrzymywało swój własny trwały stan, mimo że wszystkie używają tego samego woluminu pamięci masowej?

Używanie wielu katalogów w tym samym woluminie

Sztuczka, której moglibyśmy użyć, to używanie tego samego zasobu PersistentVolume przez wszystkie pody, ale zapewnienie każdemu podowi oddzielnego katalogu wewnątrz tego woluminu (co pokazuje rysunek 10.3).



Rysunek 10.3 Obejście problemu wspólnej pamięci masowej przez zapewnienie, że aplikacja w każdym podzie będzie używać innego katalogu plików

Jako że nie możemy skonfigurować odmiennie poszczególnych replik podu przy użyciu jednego szablonu, nie możemy nakazać każdemu wystąpieniu, jakiego katalogu powinno używać, ale możemy sprawić, aby każde automatycznie wybierało (a potencjalnie również tworzyło) katalog danych, który w danym momencie nie jest używany przez żadne inne wystąpienie. Rozwiązanie to wymaga koordynacji między wystąpieniami podów i jego poprawna realizacja nie jest łatwa. Sprawia też, że wspólny wolumin pamięci masowej staje się wąskim gardłem.

10.1.2 Zapewnianie stabilnej tożsamości każdego podu

Oprócz pamięci masowej, pewne aplikacje klastrowane wymagają również, aby każde wystąpienie miało długotrwałą, stabilną tożsamość. Pody mogą być zabijane od czasu do czasu i zastępowane nowymi. Gdy ReplicaSet zastępuje pod, tworzony jest zupełnie nowy pod z nową nazwą hosta i adresem IP, choć dane w jego woluminie mogą być tymi samymi co zabitego podu. Dla pewnych aplikacji rozpoczynanie z danymi starego wystąpienia, ale z zupełnie nową tożsamością sieciową może powodować problemy.

Dlaczego pewne aplikacje wymagają stabilnej tożsamości sieciowej? Wymaganie jest to całkiem powszechne w rozproszonych aplikacjach z pamięcią stanu (stanowych). Aplikacje takie wymagają od administratora wyliczenia wszystkich pozostałych członków klastra i ich adresów IP (albo nazw hostów) w pliku konfiguracyjnym każdego członka klastra. Jednak ilekroć pod jest rozmieszczany ponownie, otrzymuje zarówno nową nazwę hosta, jak i nowy adres IP, zatem cały klaster aplikacji musiałby być rekonfigurowany za każdym razem, gdy któryś z członków jest rozmieszczany ponownie.

UŻYWANIE DEDYKOWANEJ USŁUGI DLA KAŻDEGO WYSTĄPIENIA PODU

Kolejna sztuczka, której moglibyśmy użyć, aby obejść ten problem i zapewnić stabilny adres sieciowy członkom klastra, polega na utworzeniu dedykowanej usługi Kubernetes dla każdego indywidualnego członka. Ponieważ adresy IP usług są stabilne, moglibyśmy w konfiguracji odwoływać się do każdego członka przez adres IP jego usługi (a nie adres IP podu).

Jest to podobne do opisanego wcześniej tworzenia zasobów ReplicaSet dla każdego członka, aby zapewnić mu indywidualną pamięć masową. Połączenie tych dwóch technik prowadzi do konfiguracji pokazanej na rysunku 10.4 (pokazana jest również dodatkowa usługa obejmująca wszystkich członków klastra, gdyż zazwyczaj potrzebujemy jej na użytek klientów klastra).