# Replikacja strumieniowa w SZBD Postgres

Celem zajęć jest zapoznanie się z replikacją strumieniową w SZBD Postgres w konfiguracji z rozszerzeniem Citus w celu zwiększenia niezawodności rozwiązania.

## Konfiguracja środowiska

Rozszerzenie Citus od wersji 11 nie oferuje własnych mechanizmów do zwiększenia niezawodności bazy danych koordynatora oraz baz danych węzłów roboczych, należy skorzystać z innych rozwiązań. Rekomendowanym przez twórców rozszerzenia Citus rozwiązaniem jest [replikacja strumieniowa](https://www.postgresql.org/docs/current/static/warm-standby.html#STREAMING-REPLICATION) dostarczana bezpośrednio przez Postgresql. Rozwiązanie to opiera się na dwóch rodzajach baz danych: na podstawowej bazie danych (ang. primary database) i na czuwających bazach danych (ang. standby database). Jedna podstawowa baza danych może być replikowana do jednej lub wielu czuwających baz danych. Podstawowa baza danych jest bazą danych działającą w trybie odczyt-zapis, natomiast czuwające bazy danych działają w trybie tylko-do-odczytu. Replikacja zmian wprowadzonych w podstawowej bazie danych do czuwających baz danych odbywa się przez transferowanie przez sieć zmian zarejestrowanych w [dzienniku](https://en.wikipedia.org/wiki/Transaction_log) podstawowej bazy danych. Przetransferowane zmiany do czuwających baz danych są przez nie wykorzystywane do uspójnienia się ze stanem podstawowej bazy danych. W przypadku wystąpienia awarii podstawowej bazy danych następuje ręczne lub automatyczne przełączenie jednej z czuwających baz danych w tryb podstawowej bazy danych (ang. failover). Istnieje możliwość zamiany ról miedzy podstawową i czuwającą bazą danych w trakcie bezawaryjnej pracy system w celu np. wykonanie uaktualnienia oprogramowania (ang. switchover). W Postgres replikacja strumieniowa może odbywać się synchronicznie lub asynchronicznie. Replikacja synchroniczna wymaga aby przed zatwierdzeniem transakcji w podstawowej bazie danych zmiany tej transakcji zostały odebrane lub zastosowane przez repliki. Replikacja asynchroniczna nie stawia takiego wymogu.

W celu ułatwienia konfiguracji podstawowej i czuwającej bazy danych oraz automatycznego przełączenia w przypadku awarii podstawowej bazy danych wykorzystamy działający w oparciu o asynchroniczną replikację strumieniową operator [Kubegres](https://www.kubegres.io/) klastra Kubernetes. Operator klastra Kubernetes jest rodzajem kontrolera, który rozszerza API Kubernetes w celu zarządzania złożonymi aplikacjami stanowymi.

Pojęcie klaster jest używane w wielu kontekstach, poniższe zestawienie ma na celu uporządkowania tych kontekstów:

Klaster Postgres - zbiór baz danych Postgres działających na tym samym węźle. W związku z tym, że typowe kontenery Postgres są skonfigurowane z jedną bazą danych, to pojęcia tego nie używamy w tutorialach.

Klaster Kubegres – zbiór replikowanych baz danych Postgres, w tej replikacji dążymy aby wszystkie bazy danych w takim klastrze miały taką samą zawartość.

Klaster Citus – zbiór baz danych Postgres, który umożliwia replikację poziomą. W celu zwiększenia jego niezawodności, każdy element klastra Citus będzie klastrem Kubernetes.

Klaster Kubernetes – środowisko orkiestracji skonteneryzowanych aplikacji, potencjalnie może zarządzać wieloma klastrami Citus i klastrami innych aplikacji.

1. Niestety Kubegres nie można wdrożyć dla działającego systemu, z tego powodu w terminalu pomocniczym usuń obiekty wdrożone za pomocą pliku manifestów wykonując poniższe polecenie:

kubectl delete -f rbd-citus.yaml

1. Trwałe woluminy nie są usuwane kaskadowo wraz z usunięciem StatefulSet, które je wykorzystywały, z tego powodu wykonaj w terminalu pomocniczym poniższe polecenia w celu ich usunięcia:

kubectl delete pvc pgsql-rbd-citus-disk-pgsql-citus-sts-0

kubectl delete pvc pgsql-rbd-citus-disk-pgsql-citus-sts-1

kubectl delete pvc pgsql-rbd-citus-disk-pgsql-citus-sts-2

kubectl delete pvc pgsql-rbd-citus-disk-pgsql-citus-sts-3

1. Zainstaluj operator Kubegres wykonując w terminalu pomocniczym poniższe polecenie:

kubectl apply -f \

https://raw.githubusercontent.com/reactive-tech/kubegres/v1.19/kubegres.yaml

1. Zainstaluj mapę konfiguracyjną (ang. configuration map) wykorzystywaną przez Kubegres, wykorzystaj w terminalu pomocniczym poniży potok poleceń:

curl \

https://raw.githubusercontent.com/reactive-tech/kubegres/refs/heads/main/internal/controller/spec/template/yaml/BaseConfigMapTemplate.yaml | \

sed "s/md5/trust/g" | \

sed "/reject/d" | \

sed "/postgres.conf: |/ a \ \ \ \ shared\_preload\_libraries='citus'" | \

kubectl apply -f -

Mapa konfiguracyjna Kubernetes umożliwia składowanie i zarządzanie danymi konfiguracyjnymi, które są dostępne z poziomu Pod w postaci wartości zmiennych środowiskowych lub zawartości plików. Zainstalowana mapa definiuje między innymi zawartość 2 plików konfigurujących bazę danych Postgres uruchomianą w Pod: postgres.conf oraz pg\_hba.conf. Ten pierwszy plik jest ogólnym plikiem konfigurującym Postgres natomiast ten drugi służy do zdefiniowania sposobu uwierzytelnienia dostępu do systemu Postgres. W powyższym poleceniu zastosowano polecenia sed w celu modyfikacji zawartości obu plików do naszych celów. Pierwsze polecenie sed służy do wyłączenia w pliku pg\_hba.conf uwierzytelnienia, połączenia do bazy danych będzie można wykonywać bez podania jakichkolwiek danych uwierzytelniających np. hasła. Takie rozwiązanie ułatwi komunikację w klastrze Citus, ponieważ jego wersja Community uniemożliwia użycie portfela z hasłami. W systemie produkcyjnym można zastosować plik [.pgpass](https://www.postgresql.org/docs/current/libpq-pgpass.html) do składowania haseł lub [uwierzytelnienie za pomocą certyfikatów](https://www.postgresql.org/docs/current/auth-cert.html). Drugie polecenie sed usuwa wpis w pg\_hba.conf, który uniemożliwia zdalne połączenie z wykorzystaniem protokołu [IPv4](https://pl.wikipedia.org/wiki/IPv4). Trzecie polecenie sed dodaje w pliku postgres.conf wiersz, który umożliwia załadowanie pliku biblioteki citus.so, która jest niezbędna do działania rozszerzenia Citus. Zachęcam do samodzielnego przestudiowania zawartości źródłowego pliku YAML.

1. W terminalu pomocniczym pobierz plik manifestów, który wykorzystamy do definicji koordynatora klastra Citus, uruchom poniższe polecenie:

wget https://www.cs.put.poznan.pl/jjezierski/RBDv2/rbd-citus-coord.yaml

1. Przeglądnij zawartość pobranego pliku uruchamiając w terminalu pomocniczym następujące polecenie:

less rbd-citus-coord.yaml

Plik manifestów zawiera 2 manifesty. Pierwszy manifest opisuje parametry poufne (ang. secrets), parametry te są bardzo podobne do mapy konfiguracyjnej, z tą różnicą, że wartości parametrów poufnych są składowane w repozytorium Kubernetes w postaci zaciemnionej. W naszym przypadku parametry poufne zostały wykorzystane do przekazania hasła użytkownika *postgres*, który jest administratorem systemu Postgres oraz hasła użytkownika *replication*, który jest wykorzystywany do uwierzytelniania połączenia stosowanego do replikacji strumieniowej.

Drugi manifest służy do utworzenia z pomocą operatora Kubegres obiektów Kubernetes, które będą implementować strumieniową replikację między co najmniej dwoma bazami danych Postgres. Klucz *apiVersion* służy do wskazania wersji API obiektu Kubernetes, w naszym przypadku jest to aktualna wersja operatora Kubegres. Klucz *kind* określa rodzaj obiektu Kubernetes, wartość klucza wskazuje operator Kubegres. Klucz *spec.replicas* definiuje liczbę replik, wartość 2 oznacza, że zostaną utworzone dwa StatefulSet. Jeden StatefulSet będzie obsługiwał podstawową bazę danych, natomiast drugi będzie obsługiwał czuwająca bazę danych. Większa liczba replik zwiększy liczbę StatefulSet, które obsługiwałyby kolejne czuwające bazy danych. Klucz *spec.image* wskazuje na obraz kontenera uruchamianego w StatefulSet, wykorzystujemy obraz Postgres z zainstalowanymi plikami bibliotek rozszerzenia Citus. Klucz *spec.database* definiuje rozmiar trwałego woluminu, który zostanie wykorzystany do składowania danych bazy danych Postgres oraz miejsce jego zamontowania. Klucz *spec.env* służy do zdefiniowania zmiennych środowiskowych, których wartości są wykorzystywane przez skrypty inicjalizacyjne kontenera. Zwróć uwagę, że wartości zmiennych środowiskowych POSTGRES\_PASSWORD oraz POSTGRES\_REPLICATION\_PASSWORD zostały określone z wykorzystaniem parametrów poufnych zdefiniowanych w pierwszym manifeście.

Opuść program *less* wybierając przycisk *q*.

1. Otwórz nową zakładkę terminala i nazwij ją *diagnostyczny*.
2. Uruchom w terminalu diagnostycznym poniższe polecenie w celu monitorowania zdarzeń mających miejsce się w klastrze Kubernetes:

kubectl get events -o custom-columns=LastSeen:.lastTimestamp,From:.involvedObject.name,Reason:.reason --watch

1. Wykonaj wdrożenie pliku manifestu wykonując w terminalu pomocniczym poniższe polecenie:

kubectl apply -f rbd-citus-coord.yaml

1. W terminalu pomocniczym wydaj poniższe polecenie w celu monitorowania tworzenia stanowych Pod:

kubectl get sts -- watch

Poczekaj aż oba Pod zastaną uruchomione. Przerwij polecenie. Zapoznaj się w terminalu diagnostyczny, ze szczegółami tworzenia Pod. Zanotuj wynik. [Raport]

1. Oprócz obiektów StatefulSet zostały utworzone również bezgłowe usługi, których zadaniem jest udostępnienie nazw domenowych Pod. Usługa, która obsługuje podstawową bazę danych będzie się nazywać tak jak nazwa instancji operatora Kubegres, czyli w naszym przypadku *rbd-citus-coord*, natomiast obsługująca repliki będzie się nazywać *rbd-citus-coord-replica*. W przypadku awarii podstawowej bazy danych jedna z replik zostanie promowana przez operator Kubegres do roli podstawowej bazy danych i nazwa *rbd-citus-coord* zacznie automatycznie wskazywać na nową podstawową bazę danych. Te bezgłowe usługi nie wynikają wprost z manifestu, są one specyficzną własnością operatora Kubegres. W celu sprawdzenia utworzenia odpowiednich bezgłowych usług wydaj w terminalu pomocniczym poniższe polecenie:

kubectl get svc

Aby sprawdzić jakie punkt końcowe są skojarzone z tymi usługami użyj w tym samym terminalu polecenia:

kubectl get endpoints

Zanotuj wynik. [Raport]

1. Przygotuj manifest dla pierwszego węzła roboczego:

* W terminalu pomocniczym skopiuj manifest rbd-citus-coord.yaml

cp rbd-citus-coord.yaml rbd-citus-worker1.yaml

* Użyj ulubionego edytora tekstowego do zamiany w pliku rbd-citus-worker1.yaml wartości *rbd-citus-coord* klucza *metadata.name* na wartość *rbd-citus-worker1*.
* Wykonaj wdrożenie manifestu w terminalu pomocniczym

kubectl apply -f rbd-citus-worker1.yaml

1. Na podstawie poprzedniego punktu przygotuj manifest dla drugiego węzła roboczego i wykonaj wdrożenie.
2. Otwórz nową zakładkę terminala i nazwij ją *koordynator*.
3. Adresy utworzonych Pod są dostępne jedynie w klastrze Kubernetes. Można byłoby podłączyć się do baz danych, które w nich działają uruchamiając powłokę tak jak to robiliśmy wcześniej wywołując polecenie *kubectl exec*. W tym przypadku w celu przetestowania zdalnego połączenia do koordynatora uruchomimy w trybie interaktywnym dodatkowy samodzielny Pod o nazwie *psql* zawierający klienta *psql*. W tym celu w zakładce koordynator uruchom poniższe polecenie:

kubectl run -it psql --image postgres:16.0 /bin/bash

Jeżeli utracisz zakładkę możesz przyłączyć się ponownie do Pod za pomocą następującego polecenia:

kubectl attach psql -c psql -i -t

1. Po pojawieniu się znaku zachęty w terminalu koordynatora, w terminalu pomocniczym skopiuj do Pod *psql* skrypty SQL uruchamiając polecenie:

kubectl cp ~/loggers psql:/

1. W terminalu diagnostycznym upewnij się, że zakończyło się wdrożenie manifestów.
2. W terminalu koordynatora uruchom narzędzie *psql* łącząc się do podstawowej bazy danych koordynatora za pomocą następującego polecenia:

psql -h rbd-citus-coord -U postgres

1. Dokonaj konfiguracji nowego klastra Citus wykonując w bazie koordynatora następujące polecenia:

SELECT citus\_set\_coordinator\_host('rbd-citus-coord', 5432);

SELECT \* from citus\_add\_node('rbd-citus-worker1', 5432);

SELECT \* from citus\_add\_node('rbd-citus-worker2', 5432);

Zauważ, że jako koordynator jest wskazany host *rbd-citus-coord*, natomiast węzłami roboczymi klastra Citus są hosty *rbd-citus-worker1* i *rbd-citus-worker2*.

1. Utwórz obiekty użytkownika w klastrze Citus uruchamiając w bazie koordynatora poniższe polecenia:

\i /loggers/loggers.sql

oddzielnie:

SELECT create\_distributed\_table('organizations', 'or\_id');

\i /loggers/organizations.dmp

oddzielnie:

SELECT create\_distributed\_table('loggers', 'lo\_or\_id', colocate\_with => 'organizations');

\i /loggers/loggers.dmp

1. W bazie danych koordynatora sprawdź dostępność obiektów użytkownika wykonując poniższe polecenia SQL:

select \* from loggers where lo\_or\_id=261;

update loggers set lo\_description='changed' where lo\_or\_id=261;

1. W bazie danych koordynatora sprawdź status replikacji uruchomiając poniższe polecenie:

select \* from pg\_stat\_replication \x\g\x

1. Sprawdź i zanotuj adres IP bazy danych koordynatora, wykorzystaj poniższe polecenie:

SELECT inet\_server\_addr();

1. W terminalu koordynatora upuść narzędzie *psql* wydając polecenie quit.
2. W terminalu koordynatora przyłącz się do repliki (czuwającej bazy danych) koordynatora wykonując polecenie:

psql -h rbd-citus-coord-replica -U postgres

1. Sprawdź i zanotuj adres IP repliki bazy danych koordynatora, wykorzystaj poniższe polecenie:

SELECT inet\_server\_addr();

1. W replice bazy danych koordynatora sprawdź dostępność obiektów użytkownika wykonując poniższe polecenie SQL:

select \* from loggers where lo\_or\_id=261;

update loggers set lo\_description='changed' where lo\_or\_id=261;

Czy wykonanie obu poleceń się powiodło? Dlaczego? [Raport]

1. W replice bazy danych koordynatora sprawdź status replikacji uruchamiając poniższe polecenie:

select \* from pg\_stat\_wal\_receiver \x\g\xs

1. W terminalu koordynatora upuść narzędzie *psql* wydając polecenie quit.
2. W terminalu koordynatora uruchom narzędzie *psql* łącząc się do podstawowej bazy danych koordynatora za pomocą następującego polecenia:

psql -h rbd-citus-coord -U postgres

1. W terminalu pomocniczym zasymuluj awarię bazy danych koordynatora przez zmniejszenie do zera liczby replik StatefulSet, który obsługuje tę bazę danych. Wykorzystaj poniższe polecenie:

kubectl scale sts rbd-citus-coord-1 --replicas 0

1. W terminalu diagnostycznym obserwuj nowe zdarzenia, po utworzeniu nowej repliki bazy danych koordynatora przejdź do następnego punktu.
2. W bazie danych koordynatora sprawdź dostępność obiektów użytkownika wykonując poniższe polecenie SQL:

select \* from loggers where lo\_or\_id=261;

Co się stało? [Raport]

1. Ponów powyższe polecenie. Jaki jest efekt? [Raport]
2. Sprawdź i zanotuj adres IP bazy danych koordynatora, wykorzystaj poniższe polecenie:

SELECT inet\_server\_addr();

Czy adres się zmienił. Na jaki? Dlaczego? [Raport]

1. W terminalu pomocniczym sprawdź punkty końcowe:

kubectl get endpoints  
Czy punkty końcowe usług się zmieniły w porównaniu do wyniku punkt 11? Dlaczego? [Raport]

1. W terminalu pomocniczym sprawdź STS.

kubectl get sts

Czy coś zmieniło w kwestii koordynatora w porównaniu do wyniku punktu 10? Dlaczego? [Raport]

1. Przeprowadź analogiczny scenariusz symulując awarię czuwającej bazy danych, która powinna teraz działać na StatefulSet o nazwie rbd-citus-coord-3. [Raport]