# III. Replikacja poziomych fragmentów w SZBD Postgres

Celem zajęć jest zapoznanie się z replikacją poziomych fragmentów danych w SZBD Postgres z wykorzystaniem rozszerzenia Citus.

## 1. Przygotowanie środowiska

Niniejszy tutorial bazuje na wynikach poprzedniego, z tego powodu należy przywrócić uprzednio wykorzystywane środowisko. W tym celu wykonaj poniższe punkty:

1. Zaloguj się do maszyny wirtualnej jako użytkownik rbd używając hasła RBD#7102.
2. Otwórz okno terminala, który nazwiemy terminalem pomocniczym. Otwórz trzy kolejne zakładki w oknie terminala wybierając z menu *File* pozycję *New Tab*. Nazwij te zakładki nazwami kolejnych replik od `pgsql-citus-sts-0` do `pgsql-citus-sts-2`, wykorzystaj w tym celu pozycję *Set Title* z menu *Terminal*.
3. W terminalu `pgsql-citus-sts-0` wykonaj poniższe polecenie aby uruchomić powłokę w replice `pgsql-citus-sts-0`. Replika ta obsługuje węzeł koordynatora klastra Citus.
   kubectl exec -it pgsql-citus-sts-0 -- /bin/bash
4. W tym samym terminalu uruchom narzędzie *psql* w celu przyłączenia się do bazy danych koordynatora, wykorzystaj następujące polecenie:
   psql -U postgres
5. Analogicznie jak punkcie 3, w zakładkach *`pgsql-citus-sts-1`* i *`pgsql-citus-sts-2`* uruchom powłoki replik *`pgsql-citus-sts-1`* i *`pgsql-citus-sts-2.`*
6. Tak samo jak punkcie 4, w zakładkach `pgsql-citus-sts-1` i `pgsql-citus-sts-2` uruchom narzędzie *psql* w celu przyłączenia się do baz danych węzłów roboczych klastra Citus.
7. W bazie danych koordynatora usuń fragmentaryzowane poziomo tabele utworzone w ramach poprzedniego tutorialu wydając w bazie danych koordynatora polecenia:
   DROP TABLE measurements;
   DROP TABLE loggers;
   DROP TABLE logger_types;
   DROP TABLE organizations;
8. W bazach danych węzłów roboczych Citus wykorzystaj polecenie \dt aby upewnić się, że wszystkie fragmenty usuniętej tabeli również zostały usunięte.

## 2. Wykorzystanie replikacji Citus do zwiększenia niezawodności

Rozszerzenie Citus dostarcza możliwość replikacji poziomych fragmentów w celu zwiększenia niezawodności środowiska. Liczba replik każdego fragmentu jest sterowana parametrem citus.shard_replication_factor.

1. W bazie danych koordynatora ustaw parametr citus.shard_replication_factor na wartość 2 wydając poniższe polecenie:
   SET citus.shard_replication_factor = 2;

2. W bazie danych koordynatora utwórz replikowaną poziomo tabelę *organizations* wykorzystując poniższe polecenie:
   \i /data/loggers/loggers.sql
   oraz oddzielnie:
   SELECT create_distributed_table('organizations', 'or_id');
3. Załaduj do tabeli *organizations* dane wydając w bazie danych koordynatora poniższe polecenie:
   \i /data/loggers/organizations.dmp
4. W bazie danych koordynatora wydaj poniższe polecenie aby sprawdzić jakie fragmenty tabeli *organizations* znajdują się w bazie danych węzła *pgsql-citus-sts-3*:
   SELECT placementid,  shardid,
   case shardstate when 1 then 'ACTIVE' when 3 then 'INACTIVE' when 4  then 'TO_DELETE' end as status
   FROM pg_dist_shard JOIN pg_dist_shard_placement USING (shardid)
   WHERE logicalrelid='organizations'::regclass and nodename like 'pgsql-citus-sts-3%'
   ORDER by shardid;
   Załóżmy, że w pierwszym wierszu znajdują się informacje o fragmencie z identyfikatorem 102138 (shardid).  Jeżeli jest to inna wartość to zmień odpowiednio nazwę obiektu w następnych punktach.
5. W bazie danych koordynatora sprawdź w jakich węzłach klastra Citus znajduję się fragment z identyfikatorem 102138 (shardid).  Wykorzystaj poniższe polecenie:
   SELECT nodename, case shardstate when 1 then 'ACTIVE' when 3 then 'INACTIVE' when 4  then 'TO_DELETE' end as status
   FROM pg_dist_shard_placement
   WHERE shardid=102138;
6. W bazie danych węzła *pgsql-citus-sts-3* sprawdź jakie organizacje znajdują się w fragmencie organizations_102138 . W tym celu wykorzystaj poniższe polecenie:
   SELECT * FROM organizations_102138
   ORDER BY or_id;
7. Załóżmy, że w pierwszym wierszu znajdują się informacje o organizacji z identyfikatorem 1 (or_id).  Jeżeli jest to inna wartość to dostosuj odpowiednio poniższy test dostępności danych w przypadku zatrzymania części baz danych klastra.

8. W bazie danych koordynatora pozyskaj informacje o organizacji z identyfikatorem 1, w tym celu wykorzystaj poniższe polecenie:
   `SELECT * FROM organizations`
   `WHERE or_id=1;`
9. W terminalu pomocniczym zmniejsz liczbę replik *StatefulSet* do 3, operacja ta spowoduje usunięcie Pod obsługujące replikę o najwyższym identyfikatorze czyli *pgsql-citus-sts-3*. Wykorzystaj poniższe polecenie:
   `kubectl scale sts pgsql-citus-sts  --replicas 3`
   **Uwaga**: trwały wolumin wykorzystywany przez replikę *pgsql-citus-sts-3* nie został usunięty, jeżeli liczba replik zostanie w przyszłości zwiększona to zostanie on ponownie wykorzystany.
10. Powtórz punkt 8.
    Co się stało? Dlaczego? [Raport]
11. W pomocniczym terminalu zmniejsz liczbę replik *StatefulSet* do 2, operacja ta spowoduje usunięcie Pod obsługujące replikę o najwyższym identyfikatorze czyli *pgsql-citus-sts-2*. Wykorzystaj poniższe polecenie:
    `kubectl scale sts pgsql-citus-sts  --replicas 2`
12. Powtórz punkt 8. [Raport]
13. W terminalu pomocniczym zwiększ liczbę replik *StatefulSet* do 3, operacja ta spowoduje dodanie Pod obsługujące replikę o kolejnym identyfikatorze czyli *pgsql-citus-sts-2.*
    `kubectl scale sts pgsql-citus-sts  --replicas 3`
14. Powtórz punkt 8. [Raport]
15. W bazie danych koordynatora zmodyfikuj dane organizacji 1 wydając polecenie:
    `UPDATE organizations SET or_type='CLIENT' WHERE or_id=1;`
    `SELECT * FROM organizations WHERE or_id=1;`
    Co się stało? Dlaczego? [Raport]
16. W terminalu pomocniczym zwiększ liczbę replik *StatefulSet* do 4, operacja ta spowoduje dodanie Pod obsługujące replikę o kolejnym identyfikatorze czyli *pgsql-citus-sts-3.*
    `kubectl scale sts pgsql-citus-sts  --replicas 4`
17. Powtórz punkt 8. [Raport]
18. Powtórz punkt 5 aby sprawdzić status obu kopii fragmentu 102138. [Raport]
19. Użyj funkcji citus_copy_shard_placement do skopiowania aktywnego fragmentu w miejsce nieaktywnego fragmentu. Wykorzystaj informację dotyczące identyfikatora fragmentu (`shardid`) uzyskanych w wyniku zapytania z punktu 18. Wykonaj poniższe polecenie w bazie danych koordynatora.
    `SELECT citus_copy_shard_placement(102138,`
    `'pgsql-citus-sts-2.pgsql-rbd-citus', 5432,`
    `'pgsql-citus-sts-3.pgsql-rbd-citus', 5432);`
20. Czy udało się przywrócić spójność fragmentów? [Raport]

## 3. Wykorzystanie replikacji strumieniowej do zwiększenia niezawodności bazy danych koordynatora

Rozszerzenie Citus nie oferuje własnych mechanizmów do zwiększenia niezawodności bazy danych koordynatora, należy skorzystać z innych rozwiązań.

Rekomendowanym przez twórców rozszerzenia Citus rozwiązaniem jest replikacja strumieniowa 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 zmian zarejestrowanych w dzienniku 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). Mechanizmem replikacji strumieniowej można chronić nie tylko bazę danych koordynatora lecz również wszystkie pozostałe bazy danych klastra Citus. W celu ułatwienia konfiguracji podstawowej i czuwającej bazy danych oraz automatycznego przełączenia w przypadku awarii podstawowej bazy danych wykorzystamy operator Kubegres klastra Kubernetes. Operator klastra Kubernetes jest rodzajem kontrolera, który rozszerza API Kubernetes w celu zarządzania złożonymi aplikacjami stanowymi.

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`
2. 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`
3. Zaaplikuj ponownie plik z manifestami w celu utworzenia między innymi 3 replik StatefulSet z kontenerami wykorzystującymi obraz rozszerzenia Citus. Tym razem repliki będą obsługiwały jedynie węzły robocze klastra Citus, koordynator zostanie utworzony później z wykorzystaniem operatora Kubegres. W tym celu w terminalu pomocniczym wykonaj polecenie:
   `kubectl apply -f rbd-citus.yaml`
4. Zainstaluj operator Kubegres wykonując w terminalu pomocniczym poniższe polecenie:
   `kubectl apply -f \`
   `https://raw.githubusercontent.com/reactive-tech/kubegres/v1.13/kubegres.yaml`

5. 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 wersja Community uniemożliwia użycie portfela z hasłami. W systemie produkcyjnym można zastosować plik .pgpass do składowania haseł lub uwierzytelnienie za pomocą certyfikatów. Drugie polecenie sed usuwa wpis w pg_hba.conf, który uniemożliwia zdalne połączenie z wykorzystaniem protokołu 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.
6. 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
   ```
7. 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.

Oprócz obiektów StatefulSet zostanie utworzona również bezgłowa usługa, której zadaniem będzie udostępnienie nazw domenowych Pod. Pod który obsługuje podstawową bazę danych będzie się nazywać tak jak nazwa instancji operatora Kubegres, czyli w naszym przypadku *rbd-citus-coord*, natomiast obsługujący replikę będzie się nazywać *rbd-citus-coord-replica*. Usługa ta nie wynika wprost z manifestu, jest specyficzną własnością operatora Kubegres.

Opuść program *less* wybierając przycisk *q*.

8. Otwórz nową zakładkę terminala i nazwij ją *diagnostyczny*.
9. 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`
10. Wykonaj wdrożenie pliku manifestu wykonując w terminalu pomocniczym poniższe polecenie:
    `kubectl apply -f rbd-citus-coord.yaml`
11. Otwórz nową zakładkę terminala i nazwij ją *koordynator*.
12. 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 dodatkowy samodzielny Pod o nazwie *psql* zawierający klienta *psql*. W tym celu w zakładce koordynator uruchom poniższe polecenie:
    `kubectl run -it psql --image postgres:14.0 /bin/bash`
    Jeżeli utracisz zakładkę możesz przyłączyć się ponownie do Pod za pomocą następującego polecenia:
    `kubectl attach psql -c psql -i -t`
13. Po pojawieniu się znaku zachęty w terminalu koordynatora, w terminalu pomocniczym skopiuj do Pod *psql* skrypty SQL uruchamiając polecenie:
    `kubectl cp ~/loggers psql:/`
14. W terminalu pomocniczym upewnij się, że zakończyło się wdrożenie pliku manifestu.

15. W terminalu pomocniczym sprawdź jakie Pod zostały uruchomione, użyj poniższe polecenie [Raport]:
    kubectl get pods
16. W terminalu pomocniczym sprawdź jakie StatefulSet zostały uruchomione, użyj poniższego polecenia [Raport]:
    kubectl get sts
17. W terminalu koordynatora uruchom narzędzie *psql* łącząc się do podstawowej bazy danych koordynatora za pomocą następującego polecenia:
    psql -h rbd-citus-coord -U postgres
18. Dokonaj konfiguracji nowego klastra Citus wykonując w bazie koordynatora następujące polecenia:
    SET citus.shard_replication_factor = 2;
    SELECT citus_set_coordinator_host('rbd-citus-coord', 5432);
    SELECT * from citus_add_node('pgsql-citus-sts-0.pgsql-rbd-citus', 5432);
    SELECT * from citus_add_node('pgsql-citus-sts-1.pgsql-rbd-citus', 5432);
    SELECT * from citus_add_node('pgsql-citus-sts-2.pgsql-rbd-citus', 5432);
    Zauważ, że jako koordynator jest wskazany host *rbd-citus-coord*, natomiast węzłami roboczymi klastra Citus są hosty z domeny *pgsql-rbd-citus*.
19. 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
20. 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;
21. W bazie danych koordynatora sprawdź status replikacji uruchomiając poniższe polecenie:
    select * from pg_stat_replication  \x\g\x
22. Sprawdź i zanotuj adres IP bazy danych koordynatora, wykorzystaj poniższe polecenie:
    SELECT inet_server_addr();
23. W terminalu koordynatora upuść narzędzie *psql* wydając polecenie  quit.
24. W terminalu koordynatora przyłącz się do repliki (czuwającej bazy danych) koordynatora wykonując polecenie:
    psql -h rbd-citus-coord-replica -U postgres
25. Sprawdź i zanotuj adres IP repliki bazy danych koordynatora, wykorzystaj poniższe polecenie:
    SELECT inet_server_addr();
26. 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]

27. W replice bazy danych koordynatora sprawdź status replikacji uruchamiając poniższe polecenie:
    select * from pg_stat_wal_receiver \x\g\xs
28. W terminalu koordynatora upuść narzędzie *psql* wydając polecenie quit.
29. W terminalu koordynatora uruchom narzędzie *psql* łącząc się do podstawowej bazy danych koordynatora za pomocą następującego polecenia:
    psql -h rbd-citus-coord -U postgres
30. 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
31. W terminalu diagnostycznym obserwuj nowe zdarzenia, po utworzeniu nowej repliki bazy danych koordynatora przejdź do następnego punktu.
32. W bazie danych koordynatora sprawdź dostępność obiektów użytkownika wykonując poniższe polecenie SQL:
    select * from loggers where lo_or_id=261;
    Co się stało? [Raport]
33. Ponów powyższe polecenie. Jaki jest efekt? [Raport]
34. Sprawdź i zanotuj adres IP bazy danych koordynatora, wykorzystaj poniższe polecenie:
    SELECT inet_server_addr();
    Czy adres się zmienił. Na jaki? Dlaczego? [Raport]
35. Ponów polecenia z punktów 15 i 16. Czy coś zmieniło? Dlaczego? [Raport]
36. Przeprowadź analogiczny scenariusz symulując awarię czuwającej bazy danych, która powinna teraz działać na StatefulSet o nazwie rbd-citus-coord-3. [Raport]