II. Fragmentaryzacja pozioma w SZBD Postgres

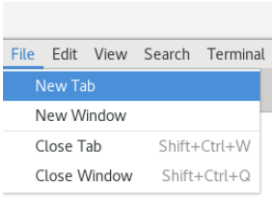
Celem zajęć jest zapoznanie się z własnościami SZBD Postgres umożliwiającymi fragmentaryzację poziomą tabel (ang. sharding).

1. Przygotowanie środowiska

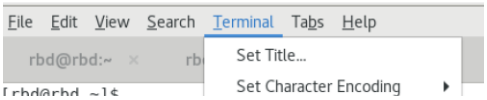
Funkcjonalność fragmentaryzacji poziomej w SZBD Postgres jest zapewniana przez rozszerzenie o nazwie Citus. Zapoznaj się z ogólnymi informacjami o tym rozszerzeniu. Rozszerzenie to jest rozwijane przez niezależną firmę, która została wykupiona przez Microsoft, w związku z tym nie jest domyślnie instalowane wraz z oprogramowaniem SZBD Postgres i należy je doinstalować lub skorzystać z gotowego obrazu. Wykorzystamy to drugie rozwiązanie. W celu przygotowania środowiska do dalszej pracy wykonaj poniższe punkty:

1. Zaloguj się do maszyny wirtualnej jako użytkownik rbd używając hasła RBD#7102.
2. Otwórz okno terminala, który nazwiemy terminalem pomocniczym.
3. Utwórz w terminalu pomocniczym klaster węzłów *k3d* z jednym serwerem oraz czterema agentami, wykorzystaj poniższe polecenie:
   ```
   k3d cluster create RBDcluster --servers 1 --agents 4 --image rancher/k3s:v1.22.2-k3s1 \
       -p "5432-5435:5432-5435@loadbalancer"
   ```
1. W pierwszym kroku pobierz plik manifestów za pomocą poniższego polecenia:
   ```
   wget www.cs.put.poznan.pl/jjezierski/RBDv2/rbd-citus.yaml
   ```
2. Otwórz plik manifestów w celu jego przeglądnięcia za pomocą polecenia:
   ```
   less rbd-citus.yaml
   ```
   Zawartość tego pliku jest zbliżona do pliku manifestów z poprzedniego tutorialu. W pierwszym manifeście, który opisuje StatefulSet za pomocą klucza *spec.template.spec.containters.image* wskazano na obraz systemu Postgres z zainstalowanym rozszerzeniem Citus. Klucz *spec.template.spec.containters.env* został wykorzystany do przekazania wartości zmiennych środowiskowych umożliwiających konfigurację kontenera. Zmienna POSTGRES_HOST_AUTH_METHOD ustawiona na wartość *trust* umożliwia wykonywanie połączeń do bazy danych bez uwierzytelnienia. Ta metoda jest niezalecana dla systemów produkcyjnych, w takich systemach należy skonfigurować uwierzytelnienie połączeń za pomocą pliku *pg_hba.conf*.
   Na końcu pliku umieszczono dodatkowy manifest, który opisuje kontroler typu bezgłowa usługa (ang. headless Service). Jego nazwa pochodzi od wartości klucza *spec.ClusterIP*, która jest ustawiona na *None*. Zadaniem tego kontrolera jest dostarczenie nazw domenowych dla wdrożonych replik *Pod*, które pasują do wartości klucza *spec.selector* tego kontrolera. Nazwy te będą dostępne wewnątrz klastra Kubernetes. Kubernetes stosuje następującą konwencję nazw domenowych *nazwa-repliki-Pod.nazwa-kontrolera.przestrzeń-nazw*.svc.cluster.local W związku z tym, że nie umieszczaliśmy naszych obiektów w żadnej przestrzeni nazw (ang. Namespace), to zostały one umieszczone w domyślnej przestrzeni nazw (*default*), przykładowa pełna nazwa domenowa pierwszej repliki będzie następująca: pgsql-citus-sts-0.pgsql-rbd-citus.default.svc.cluster.local.
   Opuść program *less* wybierając przycisk *q*.
3. Rozpocznij wdrożenie komponentów z pliku manifestów za pomocą polecenia:
   ```
   kubectl apply -f rbd-citus.yaml
   ```

4. Obserwuj postęp wdrożenia *StatefulSet* wykorzystując poniższe polecenie:
   `kubectl get sts --watch`
   Wdrożenie wymaga pobranie obrazu kontenera z repozytorium Docker, w związku z tym zajmuje chwilę. Wdrożenie zakończy się w momencie pojawienia się na terminalu wiersza, w którym liczba działających replik będzie równa liczbie żądanych replik, np.:
   `pgsql-citus-sts   3/3   1m`
   W tym momencie przerwij wykonanie polecenia wykorzystując kombinację Ctrl-c.
   Wyświetl *Pod*, które zostały utworzone do obsługi replik *StatefulSet*, wykorzystaj następujące polecenie:
   `kubectl get pods -l app=pgsql-rbd-citus`
5. Otwórz trzy kolejne zakładki w oknie terminala wybierając z menu *File* pozycję *New Tab*.



6. 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*.



7. W terminalach `pgsql-citus-sts-1` i `pgsql-citus-sts-2` będziemy wykonywać operacje na bazach danych uruchomionych w replikach `pgsql-citus-sts-1 1` i `pgsql-citus-sts-2`. Bazy te będą służyć do przechowywania fragmentów poziomych danych (ang. shard). W terminalu `pgsql-citus-sts-0` będziemy wykonywać operacje na bazie danych uruchomionej w replice `pgsql-citus-sts-0`. Baza ta będzie służyć do rozpraszania danych między różne fragmenty i koordynowania poleceń SQL kierowanych do rozproszonych poziomo fragmentów.

2. Przygotowanie schematu danych i załadowanie danych do klastra baz danych

Przykładowe dane opisują pomiary temperatury i są zgromadzone w 3 tabelach: *organizations*, *loggers*, *measurements*. Rejestratory temperatury (ang. logger) należą do określonej organizacji, zaś pomiary temperatury (ang. measurements) są zbierane przez określone rejestratory.

Dane te zostaną rozproszone w poszczególnych bazach danych tworzących klaster Citus, składającym się z jednego węzła koordynatora i wielu węzłów roboczych. Kluczowym krokiem z punkty widzenia użyteczności i wydajności mechanizmu fragmentaryzacji poziomej danych na węzły robocze jest wybór kryterium rozpraszania. Kryterium tym powinna być kolumna lub zbiór kolumn, która dzieli dane na logicznie rozłączne zbiory. W przypadku przykładowych danych jest to identyfikator organizacji. Jest to typowy wybór dla aplikacji udostępniającej usługi typu *Software as a Service*, gdzie dobrym kryterium rozpraszania jest identyfikator klienta takiej usługi. Rozszerzenie Citus wymaga denormalizacji schematu bazy danych tak aby wszystkie tabele, które będą fragmentaryzowane poziomo zawierały kryterium rozpraszania. Dodatkowo, kryterium rozpraszania powinno wchodzić w skład klucza podstawowego takich tabel.

1. W terminalu pomocniczym pobierz i rozkompresuj schemat oraz przykładowe dane:
   wget https://www.cs.put.poznan.pl/jjezierski/RBDv2/loggers.zip
   unzip loggers.zip
2. Zapoznaj się ze schematem przykładowych danych znajdującym się w pliku ~/loggers/loggers.sql. Zwróć uwagę na definicję kluczy podstawowych oraz obcych.
3. Skopiuj katalog *loggers* do katalogu */data* repliki `pgsql-citus-sts-0`.
   Użyj poniższego polecenia:
   kubectl cp loggers pgsql-citus-sts-0:/data
4. W terminalu `pgsql-citus-sts-0` wykonaj poniższe polecenie aby uruchomić powłokę w replice `pgsql-citus-sts-0`. Replika ta będzie obsługiwać węzeł koordynatora klastra Citus.
   kubectl exec -it pgsql-citus-sts-0 -- /bin/bash
5. W tym samym terminalu uruchom narzędzie *psql* w celu przyłączenia się do bazy danych koordynatora, wykorzystaj następujące polecenie:
   psql -U postgres
6. Analogicznie jak punkcie 4, w zakładkach `pgsql-citus-sts-1` i `pgsql-citus-sts-2` uruchom powłoki replik `pgsql-citus-sts-1` i `pgsql-citus-sts-2`.
7. Tak samo jak punkcie 5, 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.
8. W narzędziu *psql* repliki `pgsql-citus-sts-0` wskaż na koordynatora klastra Citus wykonując poniższe polecenie:
   SELECT citus_set_coordinator_host('pgsql-citus-sts-0.pgsql-rbd-citus', 5432);
   Zauważ, że użyto tylko początkowy fragment nazwy domenowej. Jest to możliwe dzięki temu, że Kubernetes wstrzyknął do kontenera plik /etc/resolve.conf o następującej zawartości (sprawdź to samodzielnie):
   ```
   search default.svc.cluster.local svc.cluster.local cluster.local
   nameserver …
   options ndots:5
   ```
   Za przeszukiwanie domen odpowiedzialna jest dyrektywa *search.*
9. W bazie danych koordynatora skonfiguruj klaster baz danych dołączając do niego bazy danych węzłów roboczych. W tym celu wykorzystaj poniższe polecenia:
   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);
10. Sprawdź poprawność wykonania poprzedniego kroku przez wydanie w bazie koordynatora polecenie:
    select * from citus_get_active_worker_nodes();
11. Utwórz w koordynatorze obiekty ze skryptu loggers.sql, wykonując w terminalu `pgsql-citus-sts-0` poniższe polecenie:
    \i /data/loggers/loggers.sql
12. W bazie danych koordynatora, z wykorzystaniem funkcji *create_distributed_table*, zdefiniuj fragmentaryzację tabeli *organizations*:
    SELECT create_distributed_table('organizations', 'or_id');
13. W bazie danych koordynatora, zdefiniuj fragmentaryzację tabeli *loggers*, zwróć uwagę na parametr *colocate_with*, który umożliwia rozproszenie wierszy tabeli loggers w taki sposób aby każdy wiersz tej tabeli został umieszczony w tym samym węźle co wiersz opisujący organizację, do której należy dany rejestrator:

SELECT create_distributed_table('loggers', 'lo_or_id', colocate_with => 'organizations');

14. W bazie danych koordynatora, zdefiniuj fragmentaryzację tabeli *measurements,* nie zapomnij o kolokacji z tabelą *organizations.* [Raport]
15. W bazie danych koordynatora załaduj dane do przykładowych tabel:
    \i /data/loggers/organizations.dmp
    \i /data/loggers/loggers.dmp
    \i /data/loggers/measurements.dmp
16. W bazach danych węzłów roboczych Citus sprawdź za pomocą polecenia \dt jakie zostały w nich utworzone tabele. Sprawdź schemat jednej z nich za pomocą polecenia \d nazwa_tabeli. Jak sądzisz czy wiersze tabeli *measurements* zostały równomiernie rozproszone? Dlaczego? [Raport]
17. W bazie danych koordynatora sprawdź możliwość modyfikacji wierszy rozproszonych na poszczególne bazy danych klastra:
    UPDATE loggers SET lo_description='Fridge #23' where lo_id=622 and lo_or_id=138;

3. Optymalizacja zapytań kierowanych do fragmentaryzowanych poziomo tabel

W tym punkcie zostanie sprawdzony sposób optymalizacji poleceń SQL, które są kierowane do tabel fragmentaryzowanych poziomo.

1. W bazie danych koordynatora wydaj poniższe zapytanie, zwróć uwagę, że w warunku połączeniowym znajduje się odniesienie do kryterium rozpraszania:
   select lo_description, me_time, me_temperature
   from loggers join measurements on (lo_id=me_lo_id and lo_or_id=me_or_id)
   where me_time between timestamp '2017-03-15 08:13' and timestamp '2017-03-15 09:18'
   order by me_time;
2. Sprawdź plan wykonania powyższego zapytania:
   EXPLAIN select lo_description, me_time, me_temperature
   from loggers join measurements on (lo_id=me_lo_id and lo_or_id=me_or_id)
   where me_time between timestamp '2017-03-15 08:13' and timestamp '2017-03-15 09:18'
   order by me_time;
   Na ile zadań zostało podzielone zapytanie? Jaki jest koszt wykonania jednego zadania? [Raport]
3. Wydaje, że można uzyskać wzrost wydajności wykonywanego polecenia przez utworzenie indeksu na atrybucie *me_time*. W bazie danych koordynatora wykonaj poniższe polecenie:
   CREATE INDEX me_time_idx on measurements(me_time);
4. Sprawdź czy w schematach wybranych fragmentów tabeli *measurements* w bazach węzłów roboczych został utworzony stosowny indeks. [Raport]
5. Ponownie sprawdź plan zapytania. Czy indeks został wykorzystany? Czy zmienił się koszt wykonania jednego zadania? [Raport]

6. Dodaj w warunku selekcji zapytania kryterium rozpraszania. Jaka nastąpiła zmiana w planie? [Raport]

```
EXPLAIN select lo_description, me_time, me_temperature
from loggers join measurements on (lo_id=me_lo_id and lo_or_id=me_or_id)
where me_time between timestamp '2017-03-15 08:13' and timestamp '2017-03-15 09:18'
and lo_or_id=143
order by me_time;
```

4. Tabele referencyjne

Zapoznaj się ze skryptem ~/loggers/logger_types.sql. Zawiera on definicję tabeli *logger_types*. Wiersze tej tabeli opisują rejestratory wszystkich organizacji, z tego powodu nie można jej bezpośrednio fragmentować poziomo. W związku z tym, że tabela *loggers_types* zawiera niewiele wierszy lepszym sposobem jest jej replikacja na wszystkie bazy danych klastra. Rozszerzenie Citus nazywa taki rodzaj tabel tabelami referencyjnymi. W schemacie fragmentowanej poziomo bazy danych mają on analogiczne znaczenie jak tabele wymiarów w hurtowni danych.

1. W bazie danych koordynatora uruchom skrypt /data/loggers/logger_types.sql z wykorzystaniem poniższego polecenia:
   ```
   \i /data/loggers/logger_types.sql
   ```
2. W bazie danych koordynatora wykorzystaj funkcję create_reference_table do wskazania, że tabela *loggers_types* jest tabelą referencyjną:
   ```
   SELECT create_reference_table('logger_types');
   ```
3. W bazie danych koordynatora załaduj dane do tabeli *loggers_types*:
   ```
   \i /data/loggers/logger_types.dmp
   ```
4. Wykorzystując polecenie \dt sprawdź ile fragmentów tej tabeli zostało utworzone bazach danych węzłów roboczych. Porównaj zawartość tych fragmentów z zawartością tabeli *loggers_types*. [Raport]
5. W bazie danych koordynatora zmodyfikuj nazwę jednego z typów rejestratora:
   ```
   UPDATE logger_types SET lt_name='TypRejestratora2' WHERE lt_name='LoggerType2';
   ```
6. Sprawdź czy zmiany zostały przeniesione do fragmentów w bazach danych węzłów roboczych. [Raport]
7. Sprawdź plan poniższego zapytania, które odwołuje się do tabeli *loggers_types*. Ile baz danych klastra było zaangażowanych w wykonanie tego zapytania? [Raport]

```
explain
select lt_name, lo_description, me_time, me_temperature
from loggers join measurements on (lo_id=me_lo_id and lo_or_id=me_or_id)
join logger_types on(lt_id=lo_lt_id)
where me_time between timestamp '2017-03-15 08:13' and timestamp '2017-03-15 09:18'
and lo_or_id=143
order by me_time;
```

5. Rozpraszanie połączeń do klastra Citus

1. W zakładce pgsql-citus-sts-1 spróbuj wykonać następujące polecenie:
   ```
   UPDATE loggers SET lo_description='New Fridge #23' where lo_id=622 and lo_or_id=138;
   ```

2. Domyślnie, metadanych klastra Citus są zarządzane i składowane jedynie w bazie danych koordynatora. Włącz replikację tych metadanych na węzły robocze. Zabieg ten umożliwi zlecanie zapytań i poleceń DML nie w tylko sesjach koordynatora ale również w sesjach węzłów roboczych. Polecenia DDL będą nadal obsługiwane jedynie przez sesje koordynatora. Użyj poniższe polecenia w bazie danych koordynatora:
   `SET citus.replication_model TO 'streaming';`
   `SELECT start_metadata_sync_to_node(nodename, nodeport) FROM pg_dist_node;`
3. Spróbuj powtórzyć wykonanie polecenia z punktu 1.
4. Skorzystamy z usługi *LoadBalancer* klastra Kubernetes aby przyłączyć się do jednego węzłów klastra Citus. Uruchom w terminalu pomocniczym następujące polecenie:
   `psql -h localhost -U postgres`
5. Za pomocą poniższego polecenia wydanego w terminalu pomocniczym ustal adres IP węzła klastra Kubernetes, na którym działa baza danych do której się przyłączyłeś.
   `SELECT inet_server_addr();`
6. Sprawdź nazwę domenową tego adresu. Będziesz potrzebować w tym celu narzędzia, które nie są zainstalowane w obrazie kontenera. Uruchom w trybie interaktywnym samodzielny Pod z obrazem kontenera *dnsutils*. W nowej zakładce terminala uruchom poniższe polecenie:
   ```
   kubectl run -it dnsutils \
    --image gcr.io/kubernetes-e2e-test-images/dnsutils:1.3
   ```
   Następnie w sesji z nowym Pod wykorzystaj polecenie `nslookup` do sprawdzenie nazwy domenowej adresu IP, adres IP podaj jako argument polecenia `nslookup`.
7. Otwórz nową zakładkę terminala w celu nawiązania kolejnej sesji z klastrem Citus w taki sam sposób jak w punkcie 4.
8. Sprawdź IP i nazwę domenową węzła klastra Citus, do których się przyłączyłeś.
9. Wykonaj dowolne operacje modyfikacji danych za pomocą sesji, który ustanowiłeś w punkcie 4 i 7.
10. Czy zaobserwowałeś jakieś różnice w funkcjonalności tych sesji? [Raport]

11. Dodawanie nowej bazy danych do klastra Citus

W celu wydajnego obsłużenia zwiększającej się ilości danych i zwiększającego się obciążenia systemu można dodać nową bazę danych do klastra.

1. W terminalu pomocniczym zwiększ liczbę replik *StatefulSet* do czterech. W tym celu wykonaj poniższe polecenie:
   `kubectl scale sts pgsql-citus-sts --replicas 4`
   Powyższe polecenie, zostało wykorzystane w celu poglądowym. Z punktu widzenia właściwej metodyki definicji klastra Kubernetes należałoby zmodyfikować manifest i następnie ponownie go wdrożyć.
2. W bazie danych koordynatora dodaj nową bazę danych do klastra:
   `SELECT * from citus_add_node('pgsql-citus-sts-3.pgsql-rbd-citus', 5432);`
3. Sprawdź w bazie danych nowego węzła roboczego jakie zostały utworzone fragmenty tabel. [Raport]
4. Utwórz nową tabelę i zdefiniuj dla niej fragmentaryzowanie poziome. Sprawdź, czy fragmenty tabeli zostały umieszczone we wszystkich 3 bazach danych węzłów roboczych klastra. UWAGA: przy fragmentaryzacji nowej tabeli należy przypisać parametrowi *colocate_with* wartość *none*. W

przeciwnym razie Citus będzie próbował dokonać kolokacji nowej tabeli z tabelami już istniejącymi w systemie. [Raport]