AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI
INSTYTUT INFORMATYKI

Projekt semestralny

Studium przypadku technologii Vitess

Autor: Szymon Twardosz, Dominik Pilipczuk, Krzysztof Tranquilo Dziechciarz, Wiktor Gut

Kraków, 2025

Spis treści

Rozdział 1

Cel projektu

1.1. Wprowadzenie

Celem projektu jest przedstawienie studium przypadku systemu **Vitess** – open-sourcowej platformy służącej do skalowania baz danych MySQL w środowiskach chmurowych i rozproszonych. Vitess został stworzony przez YouTube jako odpowiedź na rosnące potrzeby skalowalności i dostępności danych w dużych systemach produkcyjnych, gdzie tradycyjne podejście do baz danych relacyjnych okazywało się niewystarczające.

Vitess łączy w sobie zalety tradycyjnych baz danych (jak ACID i SQL) z elastycznością architektur opartych na mikrousługach i kontenerach. Umożliwia m.in. sharding, replikację, przełączanie awaryjne oraz zarządzanie schematem w sposób spójny i zautomatyzowany. Dzięki integracji z Kubernetesem i innymi narzędziami cloud-native, Vitess idealnie wpisuje się w potrzeby nowoczesnych, skalowalnych aplikacji.

Przykładowe firmy i usługi które korzystają z technologii Vitess to:

- **YouTube** – gdzie projekt się narodził, jako rozwiązanie problemów skalowalności bazy danych,
- **Slack** - dla obsługi ogromnej ilości wiadomości i użytkowników w czasie rzeczywistym,
- **GitHub** - do obsługi skomplikowanej infrastruktury danych przy zachowaniu wysokiej dostępności i wydajności.

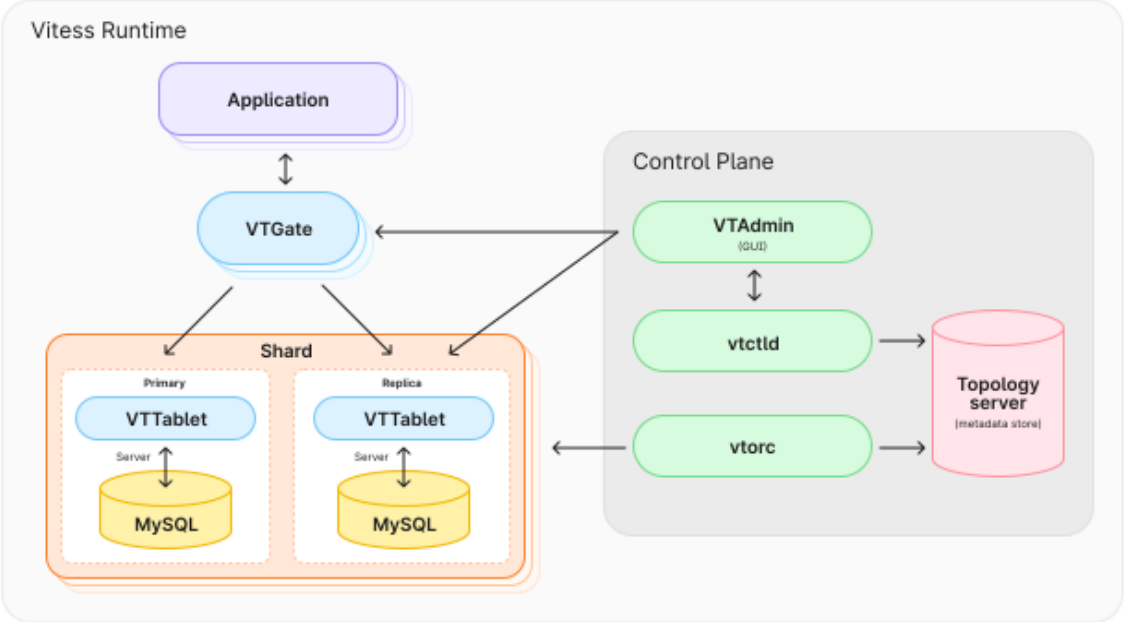
1.2. Wstęp teoretyczny

Vitess składa się z kilku kluczowych komponentów, które współpracują, aby zapewnić wydajność, skalowalność oraz wysoką dostępność baz danych. Oto najważniejsze z nich:

- Vtorc - odpowiada za automatyczne zarządzanie topologią replikacji MySQL oraz wykrywanie i reagowanie na awarie w klastrze bazodanowym.
- VTGate - brama (proxy) łącząca aplikacje z systemem Vitess. Odpowiada za przyjmowanie zapytań SQL od klientów i ich kierowanie do odpowiednich shardów i replik.
- VTTablet - serwis działający obok instancji MySQL w każdej replice. Odpowiada między innymi za zarządzanie instancją MySQL oraz odpowiadanie na zapytania przychodzące z VTGate

- vtctld - interfejs administracyjny służący do zarządzania klastrem Vitess. Umożliwia między innymi monitorowanie stanu klastra czy tworzenie nowych shardów oraz replik.
- Shardy - instancję baz danych, do których kierowane są zapytania. Każdy z nich posiada własną instancję VTTablet, która jest pośrednikiem w komunikacji między instancją bazy danych a VTGate.

  Architektura tej technologi wygląda następująco (obrazek wzięty z oficjalnej dokumentacji Vitess):



Rysunek 1.1: Architektura Vitess

1.3. Opis koncepcji

W ramach projektu opartego o system Vitess, zaprojektowano i przeanalizowano dwa scenariusze ilustrujące różne aspekty działania i odporności systemu bazodanowego opartego o sharding oraz replikację. Poniżej przedstawiono szczegółowy opis obu scenariuszy wraz z ich logicznymi krokami.

1.3.1. Sharding horyzontalny – instancje baz danych posiadają swoje tabele

Sharding horyzontalny (ang. *horizontal sharding*) polega na podziale danych tej samej tabeli pomiędzy różne instancje baz danych, zwykle na podstawie zakresu wartości klucza głównego (np. `customer_id`). Każda instancja przechowuje ten sam schemat, lecz inny podzbiór danych.

Kroki scenariusza:

1. Użytkownik wykonuje zapytanie, np. `SELECT * FROM customers WHERE customer_id = 1350`.

2. Komponent **VTGate** odbiera zapytanie i na podstawie klucza dzielącego kieruje je do odpowiedniego sharda.
3. VTGate przekazuje zapytanie do właściwego **VTTablet**, obsługującego shard zawierający dane klientów.
4. VTTablet komunikuje się z backendem MySQL i otrzymuje wynik zapytania.
5. VTGate, w razie potrzeby, agreguje wyniki i zwraca je użytkownikowi.

1.3.2. Sharding wertykalny – repliki całej bazy danych

Sharding wertykalny (ang. *vertical sharding*) polega na logicznym podziale schematu bazy danych, gdzie różne tabele są rozmieszczone na różnych instancjach. Każda instancja może posiadać swoje repliki, co umożliwia równoważenie obciążenia zapytań odczytujących.

Kroki scenariusza:

1. Użytkownik wykonuje zapytanie, np. `SELECT * FROM customers`.
2. VTGate identyfikuje, że tabela `customers` znajduje się w bazie A.
3. Zapytanie jest kierowane do odpowiedniego **VTTablet**.
4. W przypadku zapytania typu `SELECT`, VTGate może skierować je do jednej z replik (nie do instancji głównej).
5. Replika MySQL odpowiada danymi.
6. VTGate przekazuje wynik użytkownikowi.
7. Jeśli zapytanie wymaga wielu tabel (np. `JOIN`), dane są zbierane z odpowiednich shardów i scala je VTGate.

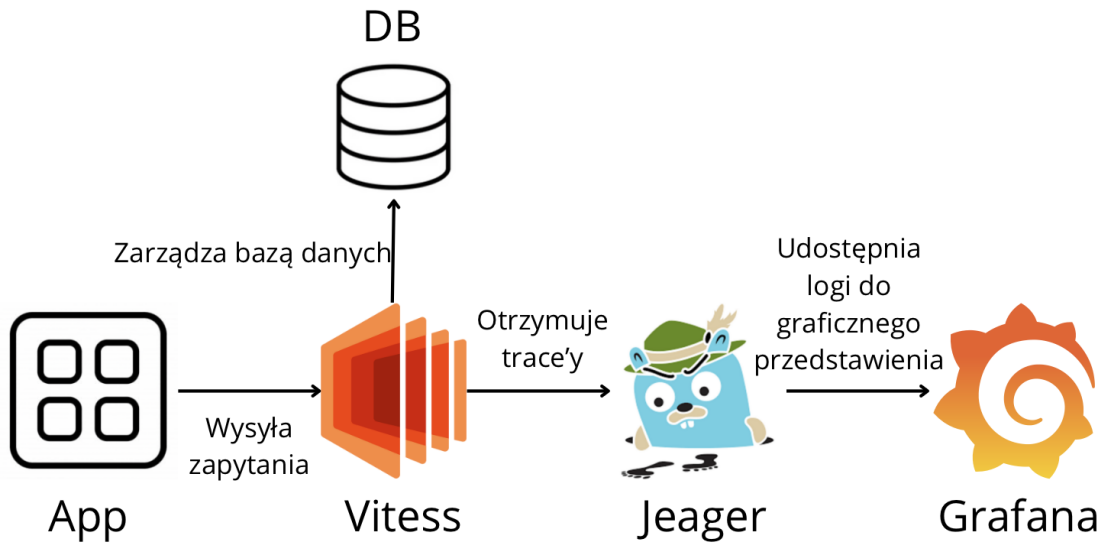
1.4. Architektura rozwiązania

W projekcie zaprezentowano uruchomienie klastra Vitess w środowisku Kubernetes przy użyciu operatora Vitess.

Cała architektura opiera się na kilku warstwach:

- **Kubernetes** – jako platforma orkiestracyjna do uruchamiania i skalowania usług kontenerowych.
- **Vitess Operator** – komponent zarządzający zasobami Vitess w klastrze K8s (shardy, replikacje, VTGate, VTTablet, itd.).
- **MySQL** – backend bazodanowy obsługiwany przez Vitess.
- **Jaeger** – system do śledzenia rozproszonych zapytań (ang. distributed tracing).
- **Grafana** – system do wizualizacji metryk.

Architektura została rozbudowana o integrację z narzędziami do obserwowalności (obserwability), dzięki czemu możliwe było przeanalizowanie rozkładu zapytań i obciążeń w czasie rzeczywistym.



Rysunek 1.2: Architektura rozwiązania

1.5. Konfiguracja środowiska

Środowisko wykonawcze projektu działa w chmurze AWS, w klastrze Kubernetes zarządzanym przez usługę Amazon EKS (Elastic Kubernetes Service). Klastrem zarządzają dwa pody uruchomione na instancjach typu t3.xlarge, które oferują 4 vCPU oraz 16 GB pamięci RAM każda. Należy pamiętać, że instalacja Vitessa na maszynach z mniejszą ilością pamięci RAM (np. poniżej 12 GB) może prowadzić do niestabilnej pracy systemu, a nawet jego awarii. Ze względu na wymagania zasobowe Vitessa, zaleca się uruchamianie go na instancjach o odpowiedniej wydajności, takich jak t3.xlarge lub wyższych.

1.6. Metody instalacji

Aby móc korzystać z wyżej opisanego demo należy wcześniej zainstalować następujące narzędzia:

- **kubectl v1.30.2** - narzędzie wiersza poleceń służące do zarządzania klastrami Kubernetes. Umożliwia wykonywanie operacji takich jak wdrażanie aplikacji, monitorowanie zasobów oraz diagnozowanie problemów w środowisku Kubernetes.
- **mysql v5.7** - narzędzie wiersza poleceń umożliwiające łączenie się z serwerem MySQL, wykonywanie zapytań SQL oraz zarządzanie bazami danych. Jest przydatne do testowania połączeń, przeglądania danych i administracji bazą.
- **vtctldclient** - narzędzie wiersza poleceń służące do komunikacji z komponentem vtctld w systemie Vitess. Umożliwia zarządzanie shardami, keyspace'ami i innymi elementami klastra Vitess.

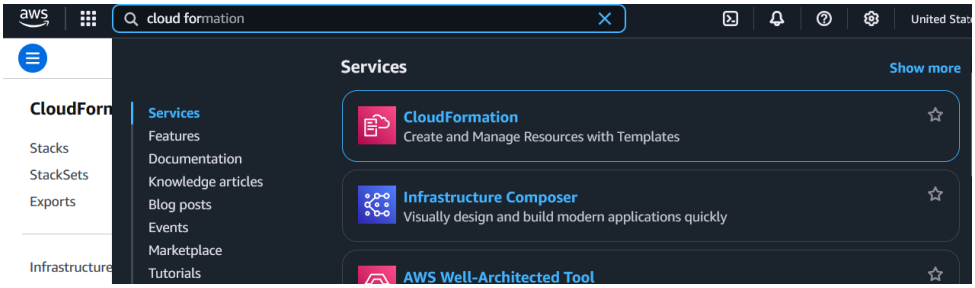
Narzędzia kubectl i mysql można zainstalować za pomocą popularnych menedżerów pakietów, takich jak brew (macOS), apt (Ubuntu) czy winget (Windows). Z kolei vtctldclient można pobrać i zainstalować przy użyciu skryptu dostępnego pod adresem: link

1.7. Jak odtworzyć projekt - krok po kroku

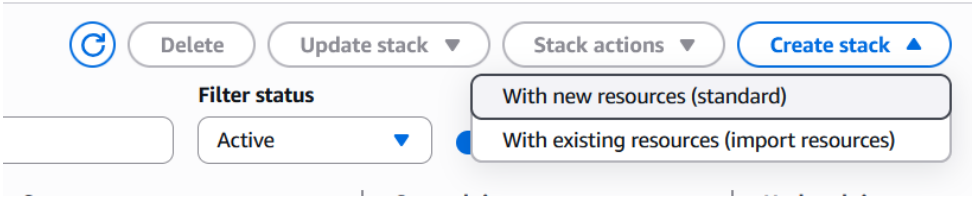
Sekcja ta ma za zadanie umożliwić innej osobie dokładne odtworzenie środowiska od zera, w tym instalacji narzędzi i ich konfiguracji. Przedstawia pełny „przepis” krok po kroku.

1.7.1. Konfiguracja środowiska AWS

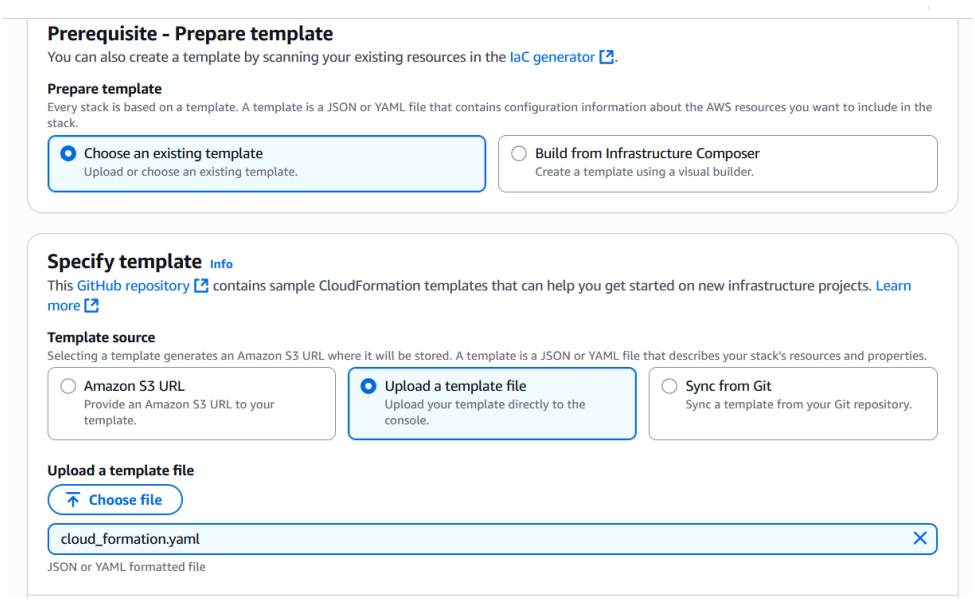
- Krok 1 - Zaloguj się do środowiska AWS oraz przejdź do zakładki AWS Cloud Formation



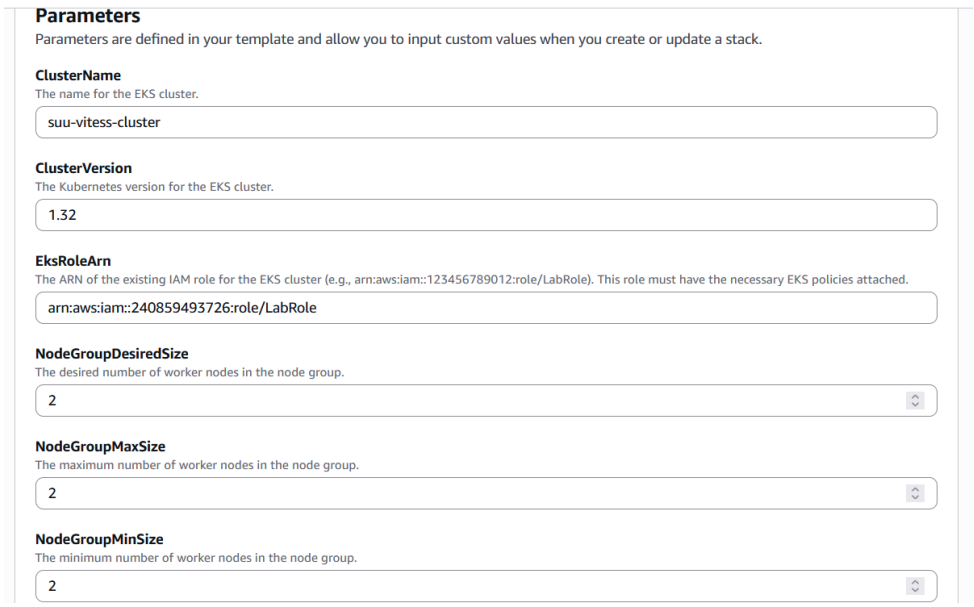
- Krok 2 - Utwórz nowy stack przy użyciu nowych zasobów klikając poniższy przycisk

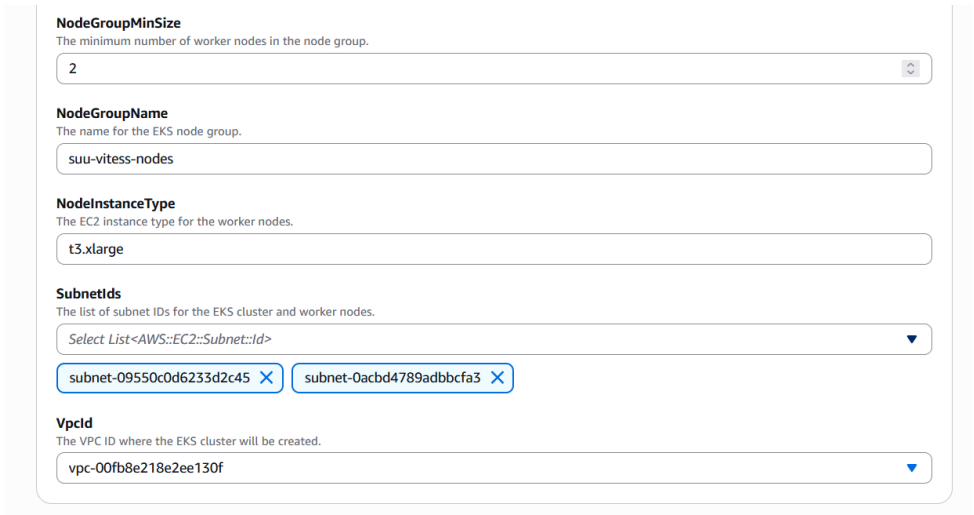


- Krok 3 - Dodaj plik cloud_formation.yaml z folderu vitess/cloud_formation jako templatkę opisującą infrastrukturę potrzebną do uruchomienia dema



- Krok 4 - Skonfiguruj pozostałe pola w następujący sposób





- Krok 5 - konfiguracja narzędzia kubectl:

```
aws eks --region us-east-1 update-kubeconfig --name suu-vitess-cluster

```

1.7.2. Konfiguracja narzędzia Jaeger

```
kubectl apply -f vitess/vitess_config/yaml/jaeger.yaml
```

1.7.3. Konfiguracja clustra Vitess

- Krok 1 - Utworzenie namespace example

```
kubectl create namespace example

```

- Krok 2 - Utworzenie Operatora

```
kubectl apply -f vitess/vitess_config/yaml/operator.yaml

```

- Krok 3 - Utworzenie początkowego clustra

```
kubectl apply -f vitess/vitess_config/yaml/101_intial_cluster.yaml

```

- Krok 4 - Uruchomienie skryptu do port-forwardingu (w osobnym terminalu)

```
        source vitess/vitess_config/pf.sh

```

- Krok 5 - utworzenie schematu bazy danych i dodanie przykładowych danych

```
vtctldclient ApplySchema --sql="$(cat
    vitess_config/sql/create_commerce_schema.sql)" commerce
vtctldclient ApplyVSchema --vschema="$(cat
    vitess_config/json/vschema_commerce_initial.json)" commerce
mysql < vitess_config/sql/insert_commerce_data.sql
```

- Krok 6 - Dodanie shardingu wertykalnego

```
kubectl apply -f vitess_config/yaml/201_customer_tablets.yaml
vtctldclient MoveTables --workflow commerce2customer
    --target-keyspace customer create --source-keyspace commerce
    --tables "customer,corder"
vtctldclient vdiff --workflow commerce2customer --target-keyspace
    customer create
vtctldclient vdiff --workflow commerce2customer --target-keyspace
    customer show last
vtctldclient MoveTables --workflow commerce2customer
    --target-keyspace customer switchtraffic --tablet-types
    "rdonly,replica"
vtctldclient MoveTables --workflow commerce2customer
    --target-keyspace customer switchtraffic --tablet-types primary
```

```
vtctldclient MoveTables --workflow commerce2customer
    --target-keyspace customer complete
```

- Krok 7 - Dodanie shardingu horyzontalnego

```

vtctldclient ApplySchema --sql="$(cat
    vitess_config/sql/create_commerce_seq.sql)" commerce
vtctldclient ApplyVSchema --vschema="$(cat
    vitess_config/json/vschema_commerce_seq.json)" commerce
vtctldclient ApplySchema --sql="$(cat
    vitess_config/sql/create_customer_sharded.sql)" customer
vtctldclient ApplyVSchema --vschema="$(cat
    vitess_config/json/vschema_customer_sharded.json)" customer
kubectl apply -f vitess_config/yaml/302_new_shards.yaml

vtctldclient Reshard --workflow cust2cust --target-keyspace customer
    create --source-shards '-' --target-shards '-80,80-'

vtctldclient vdiff --workflow cust2cust --target-keyspace customer
    create
vtctldclient vdiff --workflow cust2cust --target-keyspace customer
    show last

vtctldclient Reshard --workflow cust2cust --target-keyspace customer
    switchtraffic --tablet-types "rdonly,replica"
vtctldclient Reshard --workflow cust2cust --target-keyspace customer
    switchtraffic --tablet-types primary

vtctldclient Reshard --workflow cust2cust --target-keyspace customer
    complete

```