Wrocław, 11.05.2020

Projekt zespołowy:

Tomasz Klimczak - 61403
Justyna Stepnowska - 61676
Łukasz Rozwora - 61428

Link do aplikacji:
https://github.com/potatojesz/everyday-coelho

Cel projektu:
Przygotowanie dowolnej aplikacji korzystającej z bazy danych listy użytkowników i wysyłającej powiadomienia rss do użytkowników.

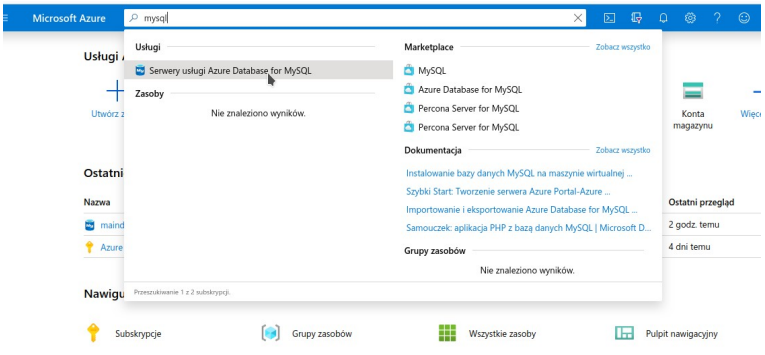
Opis aplikacji
Aplikacja wysyła do użytkownika powiadomienia mailowe o północy każdego dnia. Wysyłany jest wygenerowany cytat. Cytat zbudowany jest z pięciu zbiorów i jego konstrukcja jest losowa.

```
package everyday.coelho.model;

import com.google.common.collect.ImmutableList;

import java.util.Random;

public interface Constants {
    ImmutableList<String> START_ARTICLE = new ImmutableList.Builder<String>()
            .add("Pamiętaj, że", "Czasem nawet", "Wiedz, że", "Nie warto rozmyślać o tym, że", "Tylko", "Jedynie", "Warto rozmyślać o tym, że",
 "Zrozum:", "To oczywiste, że", "Musisz wiedzieć jedno:", "Oto cała mądrość:", "Bo miłość - tak jak", "Albowiem", "Zdarza się, że", "Słyszano, że",
 "Napisano bowiem:")
            .build();
    ImmutableList<String> NOUN = new ImmutableList.Builder<String>()
            .add("Bóg", "wojownik", "człowiek", "wszechświat", "mędrzec", "żebrak", "ludzkość", "natura", "przyjaciel", "wróg", "twój brat",
 "mistrz", "przyroda", "mnich", "nauczyciel", "uczeń", "wędrowiec", "pielgrzym")
            .build();
    ImmutableList<String> VERB = new ImmutableList.Builder<String>()
            .add("potrafi", "wie jak", "nie wie jak", "usiłuje", "jest w stanie", "zdoła", "może", "próbuje", "stara się")
            .build();
    ImmutableList<String> ARTICLE = new ImmutableList.Builder<String>()
            .add("zdobyć", "posiąść", "zyskać", "stracić", "zachować", "utracić", "zmarnować", "odzyskać")
            .build();
    ImmutableList<String> PREPOSITION = new ImmutableList.Builder<String>()
            .add("Sekret Wszechświata", "skarb", "rozum", "bogactwo", "odwagę", "miłość", "mądrość", "piękno", "dobro", "wiarę", "nadzieję",
 "prawdę", "szczęście", "spokój", "rozsądek", "sens", "moc", "siłę", "władzę")
            .build();

    Random RANDOMIZER = new Random();
```

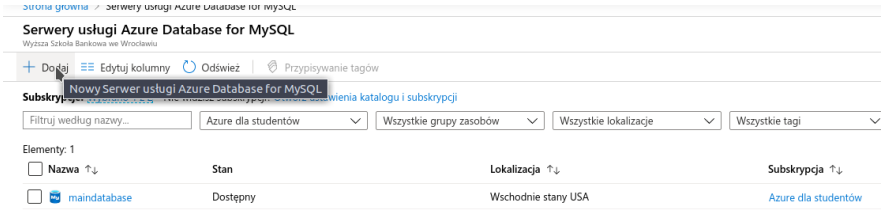
Skonfigurowanie aplikacji na azure*

a) zaloguj się do https://portal.azure.com/

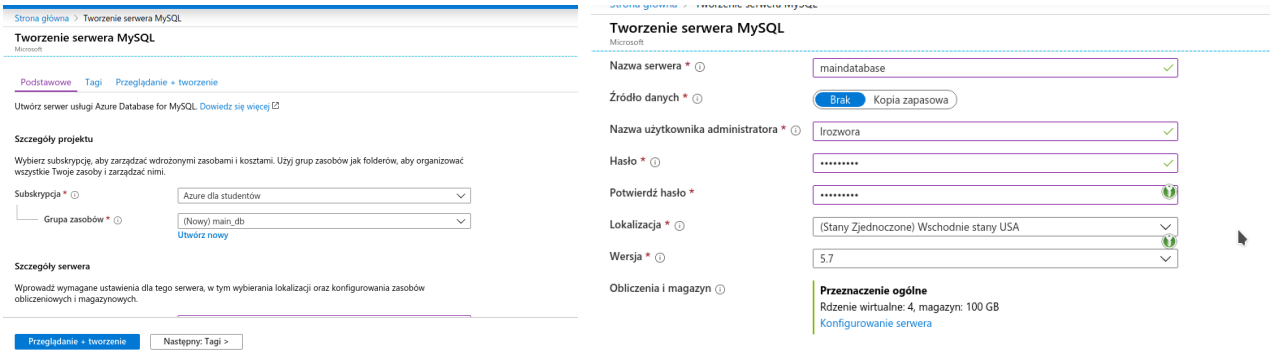
b) w pasku wyszukiwania wpisz mysql i wybierz „Serwer usługi Azure Database dla MySQL”



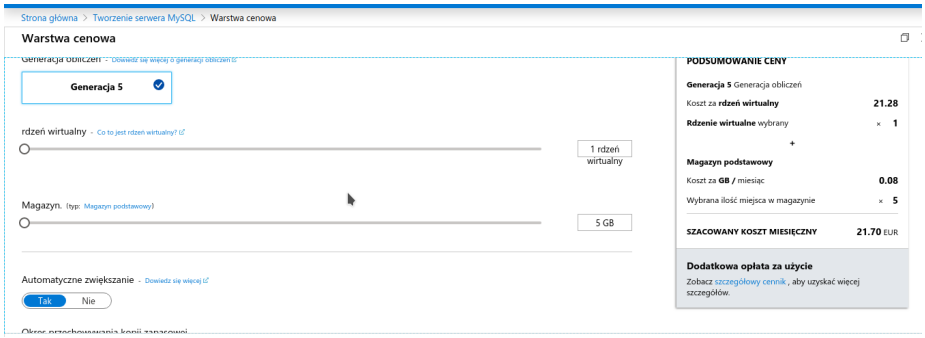
c) dodaj nowy serwer



d) uzupełnij wymagane dane i przejdź do przeglądania i tworzenia



e) W konfiguracji serwera możesz zmniejszyć ilość procesorów do 1 i przestrzeń do 5GB

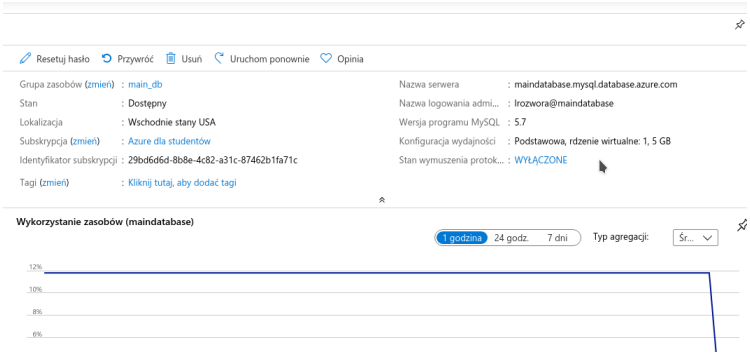


f) utwórz zasób

Stworzenie bazy danych

a) po utworzeniu serwera w prawym górnym rogu na zasobie zobaczysz swoje dane do komunikacji.

w tym wypadku jest to:
nazwa serwera: coelhodatabase.mysql.database.azure.com
nazwa użytkownika: lrozwora@coelhodatabase
hasło: nie jest widoczne, jest to hasło, które wpisałeś przy tworzeniu serwera



b) korzystając z terminala komend mysql połącz się z serwerem

> *mysql -h coelhodatabase.mysql.database.azure.com -u lrozwora@coelhodatabase – p password*

c) utwórz bazę danych „coelho_db”

> *CREATE DATABASE coelho_db*

*w screenach jest baza main, przy tworzeniu bazy tworzymy coelho_db

Instalacja aplikacji

a) pobierz aplikację

git clone https://github.com/potatojesz/everyday-coelho

b) konfiguracja danych w aplikacji

spring.datasource.url=jdbc:mysql://**coelhodatabase.mysql.database.azure.com**:3306/**coelho_db**useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC

spring.datasource.username=**root** //tutaj podaj nazwę użytkownika
spring.datasource.password=**root** //tutaj podaj hasło

# EMAIL
spring.mail.host= // serwer pocztowy
spring.mail.port= //port
spring.mail.username= //adres mail
spring.mail.password= //password
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true

c) uruchom aplikację

./mvnw spring-boot:run

Wdrażanie na Azure

a) Korzystając z Azure CLI zaloguj się do konta Azure

az login

b) W pliku pom.xml możesz zmienić nazwę i opcje.

```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
        <plugin>
            <groupId>com.microsoft.azure</groupId>
            <artifactId>azure-webapp-maven-plugin</artifactId>
            <version>1.8.0</version>
            <configuration>
                <deploymentType>jar</deploymentType>

                <appSettings>
                    <property>
                        <name>JAVA_OPTS</name>
                        <value>-Dserver.port=80</value>
                    </property>
                </appSettings>

                <resourceGroup>TestResources</resourceGroup>
                <appName>lrozworaspringbootdemo</appName>
                <region>westus2</region>

            </configuration>
        </plugin>
    </plugins>
</build>
```

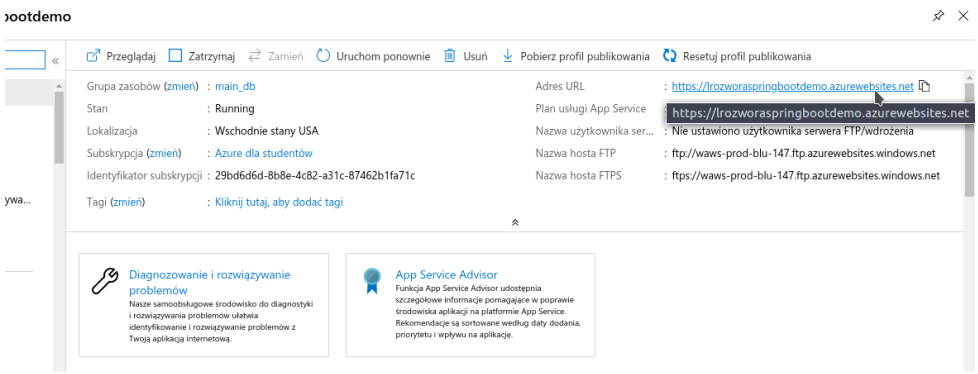
b) po zmianach odśwież aplikację

./mvnw clean package

c) umieść aplikację w app-service

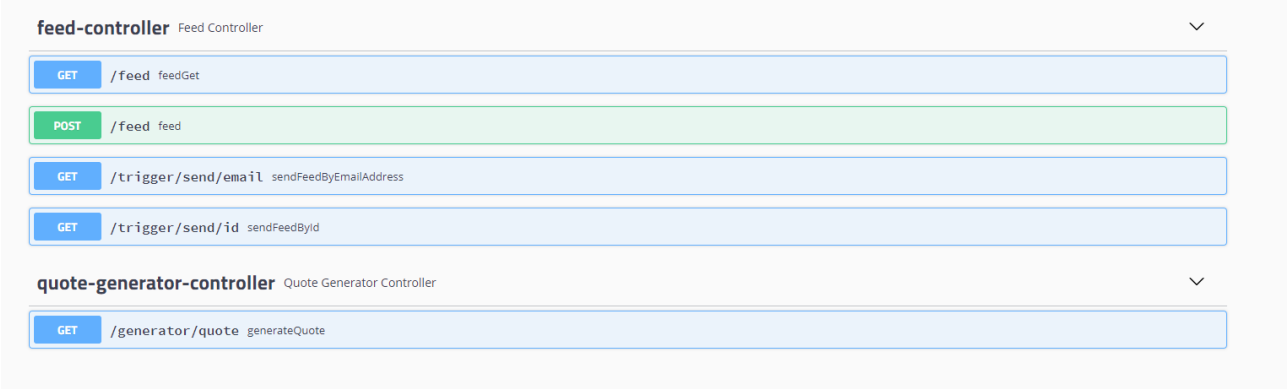
./mvnw azure-webapp:deploy

d) możesz sprawdzić na portalu czy aplikacja pojawia się w app-service



e) sprawdź czy strona działa klikając link jak w powyższym screenie

f) pod linkiem https://lrozworaspringbootdemo.azurewebsites.net/swagger-ui.html# dostępna będzie interaktywna dokumentacja rest api aplikacji utworzona automatycznie przy użyciu narzędzia swagger.



Wykorzystane technologie:

java8
mysql
spring-boot (spring-web, spring-data, spring scheduler)
hibernate
swagger
microsoft azure

Skalowanie aplikacji horyzontalnie

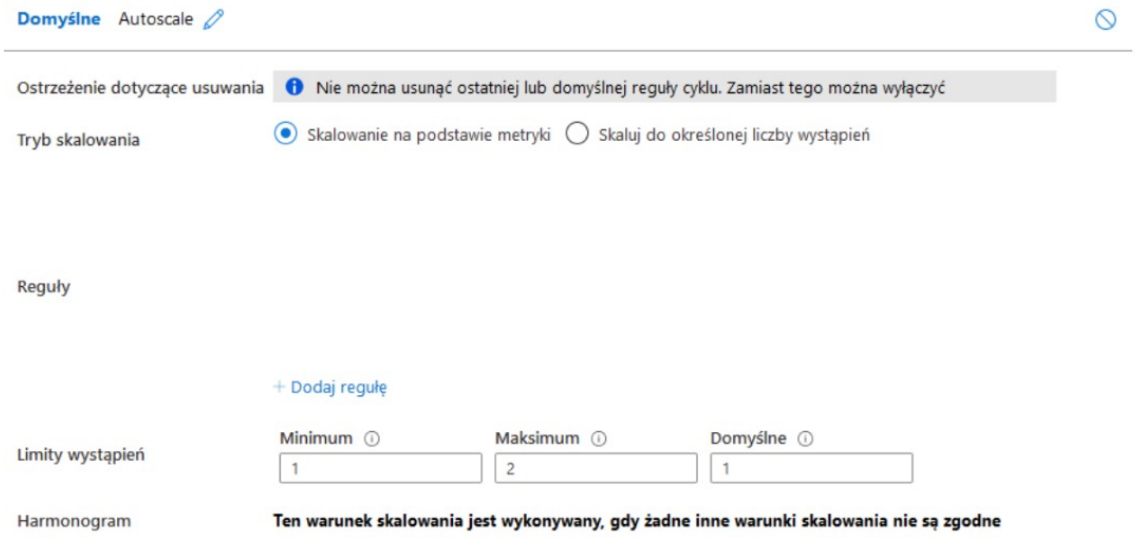
Gdy klaster zbliża się do nadmiernego użycia, skalowanie w poziomie zapewnia optymalną wydajność.

Skalowanie w poziomie nastąpi gdy:

- Liczba wystąpień klastra jest mniejsza niż maksymalna liczba wystąpień zdefiniowana przez użytkownika.
- Użycie pamięci podręcznej jest duże przez ponad godzinę.
- Procesor CPU jest wysoki przez ponad godzinę.
- Wykorzystanie pozyskiwania jest duże przez ponad godzinę.

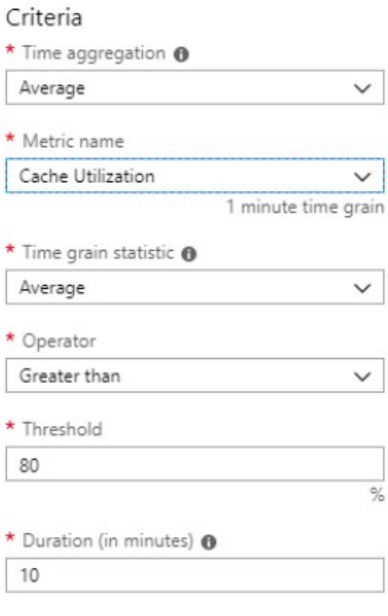
W celu skalowania aplikacji:

1. Idź do „Scale Out” i wybierz Custom autoscale
2. W scale mode wybierz „Scale based on a metric”



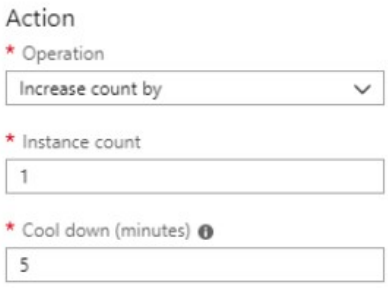
3. Dodaj reguły

Criteria:
Time aggregation – średnia
Metric name – Użycie pamięci podręcznej
Statystyka ziarna czasu – średnia
Operator – większa
Threshold - 70/80
Duration (czas na wyszukanie systemu podczas obliczania metryk) – domyślnie 10 minut



Action:
Operacja – zwiększ liczbę o
Instance count (liczbę węzłów lub wystąpień, które chcesz dodać lub usunąć po spełnieniu warunku metryki) – 1
Cool down (przedział czasu oczekiwania między operacjami skalowania) – domyślnie 5 minut

Oraz



Criteria:
Time aggregation – średnia
Metric name – Użycie pamięci podręcznej
Statystyka ziarna czasu – średnia
Operator – mniejsza
Threshold - 30
Duration (czas na wyszukanie systemu podczas obliczania metryk) – domyślnie 10 minut

Action:
Operacja – zmniejsz liczbę o
Instance count (liczbę węzłów lub wystąpień, które chcesz dodać lub usunąć po spełnieniu

warunku metryki) – 1
Cool down (przedział czasu oczekiwania między operacjami skalowania) – domyślnie 5 minut

4. W sekcji limity wystąpień wprowadź minimalną i maksymalną liczbę wystąpień, które nie będą skalowane przez klaster, niezależnie od użycia.
5. Wybierz zapisz

Podsumowanie

Dla aplikacji tego typu często dużo lepszym rozwiązaniem niż skalowanie horyzontalne, czy wertykalne, które często wiąże się z użyciem lepszego sprzętu (np. wydajniejszy serwer), czy w przypadku rozwiązań chmurowych - dużo większym kosztem w przypadku skalowania horyzontalnego.
Warto się w tym przypadku pochylić nad samą architekturą aplikacji.
Możemy zauważyć, też że sama architektura może blokować wydajność (oraz sens) wielu instancji. Jako przykład możemy się przyjrzeć naszej aplikacji, samo generowanie cytatów jest operacją bezstanową, więc bez problemu możemy odpalić n-instancji takiego serwisu nie przejmując się transakcyjnością, czy synchronizacją danych. Natomiast dużo gorzej sytuacja wygląda, gdy zobaczymy jak działa wysyłanie mailu, pobieramy wszystkie maile i wysyłamy na nie wygenerowane cytaty, przy skalowaniu horyzontalnym pojawią się problemy z synchronizacją. Musielibyśmy podzielić te dane, oraz zadbać o to żeby dla każdej skrzynki email został wysłany tylko jeden email z cytatem. Najlepszym rozwiązaniem w takiej sytuacji jest użycie jakiejś kolejki wiadomości – np. ktoraś z implementacji jms, lub jakieś nowe rozwiązanie typu kafka, czy akka. W przypadku takiej implementacji serwis od wysyłania maili, po kolei pobierałby sobie email i wysyłał na niego cytat, wtedy w zależności od tego jak szybko chcemy wysyłać maile możemy już dopasować ilość instancji takiego serwisu, czy ich wydajność.