Opis sytuacji:

Pamiętasz zadanie z początku kursu? Wracamy do naszego systemu e-commerce i projektowania aplikacji Cloud Native.

Przypomnę Ci o Twojej roli i firmie.

Jesteś architektem w firmie, która tworzy systemy w branży e-commerce dla innych firm, które sprzedają w modelu business-to-consumer w całej Europie. Systemy budujesz w oparciu o Microsoft Azure i strategia budowania systemów Cloud Native została przyjęta i jest w pełni akceptowana przez Twoich klientów

Jako architekt możesz zdecydować zarówno o architekturze systemu jak i wzorcach, które wykorzystasz. Klienci oczekują dobrze zaprojektowanego systemu, nie wnikają w Twoje decyzje.

System, który projektujesz, będzie odwiedzany przez klientów końcowych, a więc możesz się spodziewać, że:

1. W wybranych godzinach będzie zarówno dużo odwiedzin strony jak i dużo zamówień
2. System będzie miał bardzo nieprzewidywalną liczbę zamówień – zdarzą się okresy, że z systemu nikt nie będzie korzystał, ale też zdarzy się promocja typu „Black Friday”
3. Do systemu importują swoje towary również partnerzy firmy, którzy w różnych okresach roku promują wybrane produkty
4. System musi być jak najbardziej odporny na sytuacje awaryjne – jego podstawowa funkcjonalność, czyli sprzedaż i prezentacja ofert dla klientów powinna być dostępna „zawsze”

Tym razem chcemy faktycznie zbudować aplikację Cloud Native i najchętniej tak zbudować system by jak najmniej zarządzać serwerami. (pamiętasz? Serverless – Manage Your Servers Less 🙂)

Zadanie:

* 1. Wybierz 3 moduły / kawałki systemu (dowolne) może być np. System rezerwacji produktów u poddostawców, system płatności, system rezerwacji produktów w magazynie, system przetwarzania zdjęć produktów, w których wykorzystałbyś rozwiązania typu Azure Web App, Azure API Management oraz Azure Functions. Opisz ten kawałek rozwiązania (rysunki mile widziane) i min 3 punkty przy każdym wybranym module / kawałku systemu, dlaczego to jest dobry wybór.
  2. Wybierz min. 2 przykłady, w których użycie tych rozwiązań nie jest możliwe i wolisz / musisz zostać przy klasycznej maszynie wirtualnej. Napisz mi 2-3 powody, dlaczego na pewno nie da się tego zrobić.
  3. Jeśli masz już dość myślenia i planowania i nie możesz doczekać się by coś zrobić praktycznie to zadanie nr 3 jest dla Ciebie ☺ Chciałbym byś połączył kilka komponentów za sobą a przy tym nie wydał strasznej ilość budżetu. Język pisania dowolny ☺

Opis zadania:

1. Stwórz prostą funkcję, opartą o timer, która dodaje wiadomości do kolejki typu Storage Queue. Kilka przykładów tutaj: <https://docs.microsoft.com/en-us/sandbox/functions-recipes/queue-storage>
2. Stwórz drugą funkcję, która czeka na wiadomości w Storage Queue i zapisuje jej do bazy – może to być darmowa wersja Azure SQL Database
3. Stwórz trzecią funkcję, która reaguje na request http i dla podanej w parametrze daty z dokładnością do dnia (żeby było prościej) zwraca z bazy Azure SQL wszystkie komunikaty z tego dnia, które tam zostały zapisane, najlepiej w postaci JSON’a
4. Stwórz Azure API Management (kiedy będziesz miał już wszystkie inne elementy gotowe) i wystaw na Azure API Management funkcję z zadania nr 3. Dodatkowo, dodaj do wywołania tej funkcji politykę, która nie pozwala jej wywołać częściej w minucie niż trzy razy

Have fun! To zadanie pokaże Ci (mam nadzieję w prosty sposób, czym są i jak działają funkcje☺)

#TYDZIEN6.1

Aplikacja sklep e-commerce – skupiam się na funkcjonalności dotyczącej analizowania i reagowania na tzw. „porzucone koszyki” (kiedy klient wrzuca rzeczy do koszyka i finalnie nie decyduje się na zakup).

1. Web App – frontend
   1. Scale in/out w zależności od ruchu na stronie - oszczędność
   2. CI/CD
   3. Integracja z innymi usługami Azure - spójność rozwiązania
2. API Management – backend (obsługa sklepu)
   1. Obsługa autoryzacji
   2. Możliwość zarządzania wszystkimi API z jednego miejsca
   3. Ustawianie dodatkowych policy z poziomu API management
3. Functions – reakcja na porzucone koszyki
   1. wyzwalane dokładnie wtedy kiedy powinny zostać użyte (nie potrzebny jeste żaden dodatkowy proces, który będzie sprawdzał czy koszyk już został porzucony) – szybki czas reakcji, przez co zwiększamy szansę na sprzedaż
   2. Dodatkowy element systemu (nie jest to kluczowa funkcjonalność) przez co rozwój i utrzymanie może być wygodniejsze w modelu „decouple”
   3. Gdy dysponujemy statystykami dot. porzuconych koszyków w naszym sklepie, może się okazać że liczba wywołań i moc obliczeniowa potrzebna do wykonania funkcji będzie darmowa

#TYDZIEN6.2

1. Jeżeli mamy przypadek dla którego czas wykonywania funkcji może przekroczyć limity.
2. Aplikacja nie będzie wymagała skalowania, nie będzie często zmieniana – jednym słowem wszystkie zalety rozwiązań serverless i decouplingu nie są potrzebne. Wtedy możemy jedynie nałożyć sobie dodatkowy overhead, kiedy rozwiązanie najlepiej będzie funkcjonować jako monolit na jednej Vmce.

#TYDZIEN6.3

Wszystkie funkcje pisałem z poziomu Visual Studio (tylko tam umiałem dodać NuGet-em bibliotekę do połączenia z SqlDb). Całość kodu na GitHubie, dołączam screeny z wywołań 3 funkcji.



