Projektowanie i wdrażanie systemów w chmurze

Lista zadań na pracownię 2022.01.04

1. [6 pkt] Wybierz jakieś zadanie kosztowne obliczeniowo. Może być to np. kompresja danych, jakieś zadanie kryptograficzne albo coś podobnego[1]. Jeżeli na innym przedmiocie pisałeś program, który wykonuje duże obliczenia, a łatwo je podzielić na części, to jest to świetna okazja, by uruchomić go w chmurze[2]. Najlepiej, by małemu serwerowi wykonanie pojedynczego zadania zajmowało co najmniej ~10-15 sekund[3], przy pełnym wykorzystaniu CPU.

Utwórz jedną kolejkę wiadomości, której użyjemy do przechowywania informacji o oczekujących zadaniach. W tym celu możesz:

a) Skorzystać z usługi Pub/Sub w GCP, która organizuje pracę niezawodnych i szybkich kolejek[4]
b) Użyć tabeli we własnej bazie danych: każdy wiersz może reprezentować oczekujące zadanie, po wykonaniu zadania wiersz należy usunąć z tabeli.

Naszym zdaniem wariant a) jest prostszy w implementacji, ale obie opcje nagrodzimy pełnymi punktami.

Przygotuj mały program/skrypt, który oczekuje parametrów zadania w wiadomości z kolejki i gdy otrzyma wiadomość z kolejki, uruchamia wybrane przez Ciebie zadanie obliczeniowe. Niech wynik zostanie gdzieś zapisany, może np. zostać wysłany do GCS. Przygotuj obraz serwera gotowego do przetwarzania takich zapytań. Zadbaj, aby wiadomość nie była usuwana z kolejki, jeżeli obliczenia zostaną niespodziewanie przerwane.

Skonfiguruj autoskalującą grupę instancji zarządzającą takimi workerami - nadaj jej zmienny rozmiar[5]. Zadbaj, by grupa samoczynnie dobierała liczbę serwerów tak, że gdy nie ma nic do roboty, to jest mało lub zero serwerów, a gdy przybywa zadań lub obliczeń, to grupa rośnie. Jakie metryki najlepiej obserwować? Jak wybrać dobre polityki skalowania do tego scenariusza?

Przygotuj prosty skrypt (nie musisz go umieszczać na serwerze, użyj go lokalnie), który wysyła bardzo wiele wiadomości do kolejki, zlecając obliczenia, które składają się na jakąś większą całość. Upewnij się, że grupa skaluje się stabilnie i zgodnie z oczekiwaniami, a po zakończeniu obliczeń oszczędza pieniądze, zmniejszając liczbę serwerów do minimum.

2. [razem 8 pkt] Zbudujemy nieduży system udający architekturę mikroserwisów do obsługi prostej strony internetowej. Tematyka strony oraz rodzaj danych, jakie będzie przetwarzać, jest dowolna. Nie oceniamy wyglądu strony, tylko organizację infrastruktury.

Na rozgrzewkę, a zwłaszcza jeżeli nie było Cię na wykładzie, możez wykonać interaktywny samouczek o podstawach systemu Kubernetes: https://kubernetes.io/docs/tutorials/kubernetes-basics/. Przy okazji przemyśl, jak mogłaby wyglądać infrastruktura pod platformę oferującą interaktywne samouczki podobne do tych z powyższego zadania. Jakie technologie/narzędzia mogą być przydatne w budowie takiego systemu?

Z punktu widzenia użytkownika strona ma być w stanie przyjąć jakieś dane (np. zwykłym prostym formularzem + HTTP GET/POST), zapisać je do bazy danych oraz umożliwić użytkownikowi przeglądanie danych w bazie (np. prosta wyszukiwarka, lookup w bazie albo jakieś statystyki z zebranych rekordów).

Podzielmy taki system na cztery komponenty:

A) Aplikacja prezentująca stronę do użytkownika
B) Aplikacja wprowadzająca wpisy do bazy
C) Aplikacja przeprowadzająca odczyty/statystyki z danych
D) Baza danych przechowująca dane

---

[1] Odradzamy jednak łamanie sum kryptograficznych czy kopanie kryptowalut, gdyż takie obliczenia prawie zawsze naruszają regulamin usług chmurowych.
[2] A jeżeli zupełnie nie masz pomysłu, to napisz program, który rozwiązuje jakąś zagadkę brute-forcem, a jakiś parametr wejściowy określa mu, która część przestrzeni rozwiązań ma być przeszukana.
[3] Może być więcej, ale wtedy testowanie rozwiązania zajmie trochę więcej czasu. Jeśli natomiast Twoje zadanie będzie się rozwiązywać szybko, to zadbaj, by zlecić workerom wykonanie bardzo wielu takich małych tasków, aby je porządnie obciążyć.
[4] Ten poradnik zgrabnie demonstruje jak wysłać i jak odbierać wiadomości: https://cloud.google.com/pubsub/docs/quickstart-client-libraries
[5] Dla oszczędności możesz spróbować użyć serwerów typu spot / preemptible.

Do części D użyj solidnej bazy zarządzanej przez chmurę (Cloud SQL).

Części A-C będziemy uruchamiać w klastrze kontenerów. Każda z trzech aplikacji będzie spakowana w osobny obraz kontenera. Wyobrazimy sobie, że development każdego z tych trzech komponentów będzie prowadzony niezależnie (dla zabawy *możesz* napisać je w różnych językach).

[2 pkt] Napisz proste aplikacje według tego schematu oraz przygotuj je do uruchamiania w kontenerach.

Aplikacja A nie powinna komunikować się bezpośrednio z bazą. Zamiast tego powinna **wysyłać wewnętrzne zapytania HTTP** do aplikacji B oraz C. Te będą oferować wygodny interfejs dostępu do danych w bazie. Najlepiej, jeśli dodatkowo będą przeprowadzać jakieś proste przekształcenie na danych, które przekazują (np. aplikacja C może obliczać jakieś rozmaite statystyki i zwracać listę wyników).

Każdą z aplikacji nadaj reprezentatywną nazwę oraz zapakuj ją w osobny obraz Dockera. Zamieść wszystkie powstałe obrazy w Google Container Registry (GCR) lub w DockerHub. Zadbaj, aby konfiguracja aplikacji (adres/hasło do bazy, adresy do aplikacji B i C) **nie była** zawarta w obrazie, tylko pobierana ze zmiennych środowiskowych.

[4 pkt] Uruchom klaster Kubernetes w GCP. Uruchom aplikacje A-C w kontenerach na tym klastrze. Zadbaj o zwiększoną dostępność usługi: każda aplikacja powinna być uruchomiona w co najmniej trzech kopiach. Przygotuj konfigurację *deploymentów*, które pozwolą łatwo uruchomić wiele kopii aplikacji oraz wymieniać ich wersje. Przygotuj *service*, które będą udostępniać usługi na zewnątrz klastra (być może przez zintegrowany z klastrem load-balancer). Przemyśl, co powinno znajdować się w obrębie jednego *pod*a, a co w *service* (istnieje wiele poprawnych odpowiedzi) oraz jakie są zalety i wady wybranej przez Ciebie organizacji kontenerów.
Upewnij się, że całość jest dostępna dla użytkowników z publicznego Internetu, a dodatkowo, że z punktu widzenia użytkownika system funkcjonuje poprawnie.

[2 pkt] Poważnie zepsuj coś w kodzie wybranej aplikacji, nazwij to “nową wersją” i zbuduj nowy obraz Dockera. Zasymuluj release aplikacji, umieszczając nowe wersje obrazów w klastrze. Sprawdź, że klaster używa “zepsutego” obrazu. Zaobserwuj, jak zachowuje się serwis z punktu widzenia użytkownika. Za pomocą *deploymentu* wykonaj w klastrze rollback do wcześniejszej wersji.