Projektowanie i wdrażanie systemów w chmurze

Lista zadań na pracownię 2018.01.21

1. [0 pkt]
Na rozgrzewkę, zwłaszcza jeżeli nie było Cię na wykładzie, wykonaj sześcioetapowy, interaktywny samouczek o podstawach systemu Kubernetes: https://kubernetes.io/docs/tutorials/kubernetes-basics/

2. * [0 pkt extra]
Przemyśl, jak mogłaby wyglądać infrastruktura pod platformę oferującą interaktywne samouczki podobne do tych z powyższego zadania. Jakie technologie mogą być przydatne?

Zalecamy używanie Kubernetesa w GCP, gdzie jest najwygodniejszy. Nic nie stoi na przeszkodzie, a wręcz jest wskazane, by wszystkie zadania zrealizować w tym samym klastrze.

3. [8 pkt]
Zbudujemy nieduży system udający architekturę mikroserwisów do obsługi prostej strony internetowej. Tematyka strony oraz rodzaj danych, jakie będzie przetwarzać, jest dowolna. Nie interesuje nas treść ani wygląd strony, tylko organizacja infrastruktury.

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

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

Pozostałe części 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* je napisać w różnych językach).

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, jeżeli 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 nazwij nieco sensowniej oraz opakuj w osobny obraz Dockera i zamieść wszystkie powstałe obrazy w Google Container Registry (GCR) lub 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.

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 na różnych fizycznych instancjach. Przygotuj konfigurację *deploymentu* (jednego lub kilku), która pozwoli Ci łatwo uruchomić wiele kopii aplikacji oraz łatwo wymieniać ich wersje. Przygotuj *service* (jeden lub kilka), 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 użytkownika z publicznego Internetu, a dodatkowo, że system funkcjonuje poprawnie z punktu widzenia użytkownika.

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.

Szczegółowa punktacja:
- [2 pkt] Przygotowanie apek i ich obrazów
- [4 pkt] Konfiguracja kubernetesa
- [2 pkt] Aktualizacja aplikacji i rollback

4. [8 pkt] + [2 pkt extra]
Zadanie z poprzedniej pracowni, gdzie wykonywaliśmy obliczenia wywoływane kolejką, uruchomimy tym razem w klastrze Kubernetesowym. Program wykonujący obliczenia skonteneryzuj i zamieść gdzieś gotowy obraz. Następnie utwórz *deployment* który zarządza wieloma sztukami tej aplikacji.

Tym razem jako kolejki nie używaj SQSa od AWS ani PubSuba od GCP, tylko postaw własną kolejkę jako osobny *service* w klastrze, aby być mniej uzależnionym od konkretnej chmury. Możesz użyć np. RabbitMQ albo Apache Kafka (istnieją gotowe kontenery)[1]. Nie zależy nam na niezawodnym storage pod taką kolejką, przechowywane przez nią komunikaty i tak są tylko tymczasowe. Być może konieczne będzie lekkie zcustomizowanie obrazu kolejki, by ją skonfigurować na własne potrzeby.

Workery po wykonaniu obliczeń mogą wysłać plik np. do jakiegoś object storage, ale mogą też odpowiedź zawrzeć w wiadomości na innej kolejce (ale serwer obsługujący kolejki wystarczy jeden).

Wiadomości zlecające obliczenia możesz wysłać do kolejki w dowolny “ręczny” sposób (może się przydać *kubectl port-forward*), ale *jeśli masz ochotę* możesz przygotować **bardzo prosty** interfejs użytkownika “stronkę” z której można zażądać obliczeń i obejrzeć ich wynik. Naturalnie, taka stronka powinna być uruchomiona w klastrze, uczyń ją dostępną użytkownikom przez load-balancer.

Gdy upewnisz się, że całość działa, skonfiguruj HorizontalPodAutoscaler (HPA) dla deploymentu z workerami. Ustaw go tak[2], by przy braku zleconych obliczeń w klastrze było niewiele podów z workerami, a gdy jest dużo pracy do wykonania, by takich podów automatycznie przybywało.

Na koniec skonfiguruj jeszcze Cluster-Autoscaler. W GCP powinno być to bardzo proste. Ten mechanizm będzie uruchamiał dodatkowe node w klastrze, gdy zabraknie dostępnych zasobów. Upewnij się, że faktycznie tak się dzieje, tj. GCP uruchamia nowe instancje, by zmieścić liczbę workerów większą, niż oryginalnie mieściła się w klastrze, i wyłącza instancje, gdy przestają być potrzebne.

Szczegółowa punktacja:
- [2 pkt] Uruchomienie i konfiguracja własnej kolejki
- [2 pkt] Konfiguracja kubernetesa
- [2 pkt] HorizontalPodAutoscaler
- [2 pkt] Cluster-Autoscaler
- [2 pkt extra] Interfejs zlecania obliczeń i oglądania wyników

5. * [2 pkt extra]
Do zadania 3 lub 4 skonfiguruj dowolne narzędzie CI/CD (np. CircleCI lub TravisCI) tak, by opublikowanie nowych commitów w Twoim repo z kodem aplikacji skutkowało automatycznym zbudowaniem nowego obrazu, umieszczeniem go w Twoim GCR, oraz zaktualizowaniem deploymentu kubernetesa by używał nowej wersji obrazu. Upewnij się, że proces jest całkiem samoczynny, tj. że push do repozytorium skutkuje pojawieniem się nowej wersji aplikacji w klastrze.

[1] Istnieją też gotowe kontenery, które opakowują różne silniki kolejkowe w interfejs identyczny z AWS SQS.
[2] Jest bardzo wiele rozwiązań.