Projektowanie i wdrażanie systemów w chmurze

Lista zadań na pracownię 2022.01.18

Maksimum punktów do zdobycia za tę listę zadań wynosi 12 pkt + 2 pkt ekstra. Nie ma znaczenia, którymi zadaniami zdobędzie się te punkty, nie trzeba ich wykonywać po kolei.
Maksimum punktów na kolejna (ostatniej) liście zadań będzie wynosić 8 pkt - co oznacza, że łączne maksimum punktów ze wszystkich pracowni w semestrze to, zgodnie z zapowiedziami, 73 pkt.
Zalecamy zrobić wszystkie zadania w tym samym klastrze, aby zasymulować naturalne zastosowanie Kubernetesa: wiele rożnych aplikacji uruchomionych obok siebie. Zadanie 4 nie całkiem ma sens w klastrze typu "Autopilot", gdzie GCP zajmuje się skalowaniem naszego klastra, więc warto zadbać by od początku używać klastra typu "Standard".

1. [2 pkt] Uruchom dowolną platformę CMS na Kubernetesie. Jeśli jeszcze się nie znudził, możesz ponownie użyć Wordpressa[1]. **Nie** używaj bazy danych dostarczanej przez chmurę - bazę danych też zainstalujemy wewnątrz klastra[2]. Samodzielne zorganizowanie wszystkich komponentów byłoby dość skomplikowane - zamiast tego skorzystaj z "managera pakietów". Zapoznaj się z podstawami Helma i użyj go, aby zainstalować w klastrze bazę danych oraz CMS używając gotowych szablonów (helm charts) dla jednego i drugiego. Konieczne będzie skonfigurowanie parametrów do szablonów tak, by przekazać CMSowi parametry połączenia do bazy.
   Przyjrzyj się (czytając kod szablonu, lub oglądając output z odpowiedniej komendy helma), jakie dokładnie zasoby zostają dodane do klastra przy instalowaniu tych szablonów i zrozum ich znaczenie[3]. Uczyń CMSa dostępnym na zewnątrz klastra i upewnij się, że całość działa.
   Używając helma wykonaj zmianę wersji CMSa na nowszą/starszą, sprawdź jakie zmiany zostaną wprowadzone w klastrze i przyjrzyj się zachowaniu systemu podczas takiej aktualizacji.

2. [2 pkt] W klastrze, w którym masz uruchomione przynajmniej dwie aplikacje HTTP (mogą być np. programy z poprzedniej pracowni lub zadań powyżej), przygotuj konfigurację Ingressa tak, aby obie aplikacje były dostępne przez ten sam load-balancer. Ruch do aplikacji rozdziel za pomocą ścieżek w zapytaniu HTTP lub za pomocą nazwy hosta[4]. Upewnij się, że obie aplikacje dostępne w ten sposób działają całkiem niezależnie i można je aktualizować bez wpływu na drugą, nie zmieniając konfiguracji Ingressa.

3. [3 pkt] Wybierz dowolną platformę CI[5] i połącz ją z repo, w którym trzymasz kod jakiejś własnej aplikacji uruchamianej w Kubernetesie. Przygotuj pełną konfigurację CI, która kod umieszczony w repo automatycznie wdroży w klaster. W tym celu musi składać się z co najmniej trzech etapów - testów, budowy nowego kontenera i wymiany wersji kontenera używanej w klastrze. Nie musisz pisać zaawansowanych testów aplikacji, dla celów demonstracji wystarczy coś prostego, aby tylko upewnić się, że nie ma z nią poważnych problemów. Ważne, aby deployment nie następował, gdy testy nie wykonają się poprawnie. Skonfiguruj klaster tak, by ta aplikacja była dostępna publicznie. Upewnij się, że cały proces przebiega automatycznie - tj. po `git push` poprawnego kodu nie trzeba wykonywać

[1] Realizacja tego zadania jest praktycznie niezależna od wyboru CMSa, więc może warto spróbować Drupal, Joomla, albo jeszcze coś innego?
[2] Ogólnie używanie bazy chmurowej zamiast zarządzać nią samemu w klastrze często jest świetnym pomysłem, ale w tym zadaniu chcemy się przekonać, że jak tylko chcemy, to da się mieć wszystko w klastrze.
[3] Szczególnie warto zainteresować się jak baza danych realizuje trwałe przechowywanie danych - skoro same Pody są ulotne?
[4] Nie musisz w tym celu posiadać ani kupować domeny, na potrzeby tej zabawy wystarczy, że tylko Twój komputer będzie znał "zmyśloną" domenę - wystarczy dodać wpis typu `12.34.56.78 mojadomena.com` do pliku `/etc/hosts` aby wszystkie lokalne programy łącząc się do mojadomena.com wysyłały request pod IP naszego load-balancera.
[5] np. Github Actions, GitLab CI, TravisCI, CircleCI - wiele z nich oferuje darmowe plany.

absolutnie nic więcej, by (po pewnym czasie) na publicznym endpoincie pojawiła się nowsza wersja; oraz że pushowanie zepsutego kodu (w sposób wykrywalny przez testy) nie wdroży uszkodzonej aplikacji.

4. [5 pkt] W klastrze kubernetesowym z uruchomioną aplikacją HTTP (możesz wykorzystać rozwiązanie z poprzedniej pracowni, napisać nową prostą aplikację lub wykorzystać jakiś kontener z gotowcem) przygotuj Horizontal Pod Autoscaler, który będzie kontrolował liczbę podów z aplikacją. Sprzęż go z wybraną sensowną metryką, np. zużyciem CPU tych podów, a najlepiej liczbą requestów na sekundę. Uruchom też cluster-autoscaler. W GCP jest on “managed”; zintegrowany ze sterowaniem z poziomu interfejsu chmurowego, więc nie trzeba go samodzielnie uruchamiać w klastrze.
   Wytestuj oba mechanizmy skalowania; obciąż aplikację (mogą się przydać *httperf* albo *jmeter* lub własny skrypt), zaobserwuj rosnącą liczbę podów i upewnij się, że gdy przestaną się mieścić w klastrze, automatycznie dołączą do niego nowe instancje. Następnie zatrzymaj obciążenie, i upewnij się, że wszystko wróci do początkowego stanu, tj. dodatkowe node’y oraz pody zostaną automatycznie usunięte.

5. [4 pkt] Uruchom w klastrze stos ELK, czyli kombinację trzech programów często używaną do przetwarzania logów z aplikacji: Logstash do zbierania i przekształcania logów, Elasticsearch jako baza danych do trwałego trzymania ich, oraz Kibana dostarczająca interfejs użytkownika do wyszukiwania i analizy logów. Możesz dla ułatwienia użyć Helma i gotowy chart nadający się do tego celu. Następnie skonfiguruj którąś z aplikacji w tym samym klastrze, by jej logi trafiały do ELK. Użyj Kibany by upewnić się że stają się dostępne na żywo, oraz wykonaj Kibaną jakieś nietrywialne wyszukiwanie lub wizualizację.