Inżynieria oprogramowania musi nadążać za biznesem

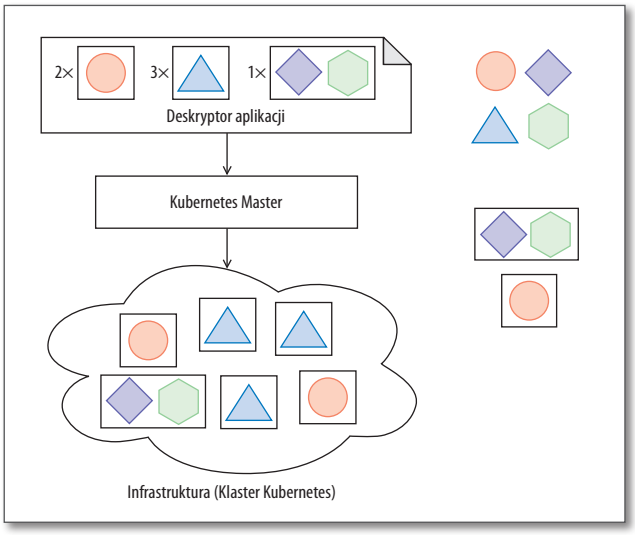
„Wyzwania inżynierii oprogramowania dla wsparcia najważniejszych trendów technologicznych" to jeden z bloków tematycznych konferencji zorganizowanej przez PTI z okazji Światowego Dnia Społeczeństwa Informacyjnego. Interesujące wnioski z wystąpień i dyskusji panelowej tego bloku przygotowali specjalnie dla „Domeny" jego uczestnicy: prof. Cezary Orłowski (dyrektor IBM Center for Advanced Studies on Campus w Wyższej Szkole Bankowej), prof. Bartosz Baliś (Instytut Informatyki Akademii Górniczo-Hutniczej, Sano Centre for Computational Medicine), Joanna Kocot (kierowniczka Laboratorium Interdyscyplinarnych Obliczeń Naukowych ACK Cyfronet AGH), Mateusz Nowak (Software Mind) i Tomasz Kulisiewicz (Sektorowa Rada ds. Kompetencji – Informatyka).

Pandemia znacząco, i wygląda na to, że trwale, zmieniła biznes i interakcje społeczne, wyznaczając nowe oczekiwania wobec informatyki. Duże firmy konsultingowe zwracają uwagę na rosnące znaczenie ekosystemów wymiany danych i automatyzacji modeli zarządzania operacjami IT. – *Automatyzacja IT na dużą skalę obejmuje trzy główne elementy: standaryzację i automatyzację infrastruktury lokalnej, aby mogła być zarządzana za pomocą kodu, standaryzację i automatyzację oprogramowania, narzędzi i systemów, aby możliwe było zarządzanie nimi za pomocą kodu oraz optymalizację automatyzacji za pomocą reguł i uczenia maszynowego* – wskazują eksperci (https://www2.deloitte.com/pl/pl/pages/technology/topics/trendy-technologiczne-deloitte.html).

Trudno też wyobrazić sobie zarządzanie urządzeniami Internetu Rzeczy bez narzędzi do automatyzacji rejestracji, konfiguracji, zdalnej i bezprzewodowej aktualizacji oprogramowania oraz monitorowania.

Konteneryzacja aplikacji

Proces wytwarzania oprogramowania, w szczególności w architekturze mikroserwisów, coraz częściej jest automatyzowany za pomocą konteneryzacji. Konteneryzacja aplikacji to technologia umożliwiająca tworzenie lekkich, przenośnych i uruchamialnych pakietów oprogramowania. Kontener zawiera pojedynczą aplikację i potrzebne jej środowisko wykonawcze, na mikroserwis składa się kompozycja kilku kontenerów, zaś złożony system jest zbiorem komunikujących się mikroserwisów. Wiodącą obecnie technologią, ułatwiającą tworzenie i uruchamianie aplikacji w infrastrukturze obliczeniowej dużej skali, jest Kubernetes[1].

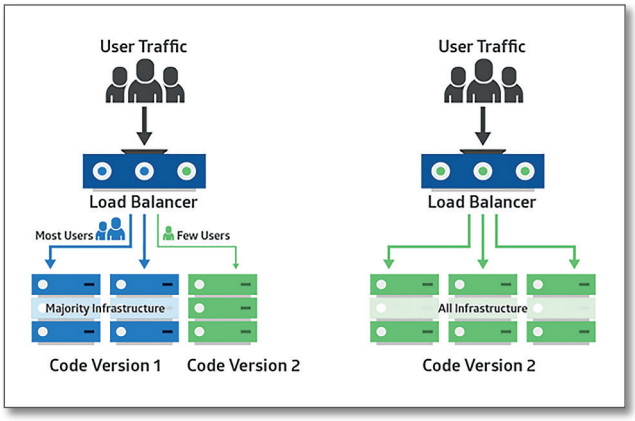


Rys. 1. Idea działania Kubernetes

Kontenery aplikacji to lekkie, przenośne i uruchamialne pakiety oprogramowania.

Pod Kubernetes to podstawowa jednostka wykonawcza w Kubernetes, która jest złożeniem jednego lub więcej kontenerów współdzielących zasoby.

Kubernetes to technologia dostarczająca warstwy abstrakcji nad infrastrukturą obliczeniową. Twórca oprogramowania tworzy Deskryptor Aplikacji i zleca jego wykonanie do Kubernetes Master (rys. 1). Kubernetes zarządza klastrem maszyn i decyduje, gdzie uruchomić poszczególne komponenty złożonej aplikacji. Kubernetes jest też platformą, która rozwiązuje lub znacznie ułatwia rozwiązywanie problemów typowych dla złożonych i rozproszonych aplikacji. Do problemów tych należą: równoważenie obciążenia, odporność na awarie, zarządzanie konfiguracją, a także automatyczne wdrażanie nowych wersji oprogramowania.



Rys. 2. Przykład procesów Kubernetes

Kubernetes jest platformą ułatwiającą zarządzanie złożonymi, rozproszonymi aplikacjami. Rys. 2 przedstawia dwa przykładowe problemy typowe dla takich aplikacji: równoważenie obciążenia oraz automatyczne wdrażanie nowej wersji oprogramowania bez przerywania działania aplikacji (tzw. *rollout*). Ze względu na duże obciążenie systemu komponenty aplikacji są zreplikowane, a ruchem zarządza System Równoważenia Obciążenia. Kubernetes stopniowo zastępuje repliki wykonujące Wersję 1 oprogramowania (kolor niebieski) nowymi (kolor zielony) (*blue-green deployment*). Możliwe też jest pilotażowe wdrożenie nowej wersji aplikacji tylko dla niewielkiej części użytkowników (tzw. wdrożenie kanarkowe – *canary deployment*). Monitorowanie działania tej części aplikacji przez pewien czas umożliwi stwierdzenie, czy nowa wersja oprogramowania działa prawidłowo. W razie potrzeby Kubernetes umożliwia automatyczne wycofanie wdrożenia i powrót do starej wersji oprogramowania (*rollback*).

1 M. Lukša, *Kubernetes w akcji*, PWN, Warszawa 2021.

> *Kubernetes jest technologią, która ułatwiła automatyzację i ustandaryzowanie procesu tworzenia i uruchamiania złożonego oprogramowania opartego na mikroserwisach, ukrywając złożoność zarządzania infrastrukturą obliczeniową.*

Wydaje się, że przyszłością są platformy bazujące na Kubernetes jako warstwie zarządzania zasobami, przykładem takiej platformy jest Knative, który realizuje paradygmat obliczeń bezserwerowych z wykorzystaniem Kubernetes.

CI i CD z pomocą

Ciągła integracja (CI – *Continuous Integration*) oraz Ciągłe dostarczanie (CD – *Continuous Deployment, Continuous Delivery*) to jedne z kluczowych praktyk, wprowadzających automatyzację do procesu dostarczania oprogramowania. Ciągła integracja zakłada scalanie każdej zmiany w kodzie z bazą (kodu), z której tworzony jest produkt – deweloperzy pracują na tym samym kodzie, dodając do niego względnie niewielkie elementy. Zapobiega to tworzeniu równoległych wersji kodu, które przed wdrożeniem należy zintegrować. Po dołączeniu każdej zmiany, automatycznie uruchamiane są testy, dzięki czemu większość błędów od razu zostaje wychwycona i poprawiona.

Ciągłe dostarczanie rozszerza koncept ciągłej integracji o każdorazowe budowanie i wdrażanie aplikacji po zmianie środowiska, co sprawia, że tworzenie kolejnych wersji oprogramowania staje się dużo łatwiejsze – wymaga mniejszego zaangażowania pracowników i zabiera dużo mniej czasu, zatem pozwala na częstsze wprowadzanie nowych wersji, a tym samym na szybszą, dotyczącą nowych funkcjonalności, informację zwrotną od klienta czy użytkowników.

Procesy CI oraz CD zakładają automatyzację procesu wdrożenia, która najczęściej realizowana jest w formie potoków (*pipeline*), wykonujących po kolei elementy procesu. Potok podzielony jest zazwyczaj na etapy, np. „budowanie", „testowanie", składające się z mniejszych zadań. O ile etapy zazwyczaj wykonywane są jeden po drugim, z możliwością zdefiniowania warunków wykonania danego etapu, o tyle już wewnętrzne zadania (które mogą składać się z jeszcze mniejszych kroków) często mogą być wykonywane równolegle na różnych agentach. Potok może być wyzwalany dowolnie zdefiniowaną akcją, np. opublikowaniem nowej wersji kodu aplikacji.

Wiele z istniejących narzędzi oferujących rozwiązania potokowe integruje się z repozytoriami kontroli wersji, przeważnie bazującymi na systemie Git (https://git-scm.com/), co umożliwia automatyczne uruchomienie potoku, np. w momencie integracji nowej zmiany do głównej części kodu lub dowolnej innej akcji na repozytorium kontroli wersji. Repozytoria kontroli wersji wyposażone w mechanizm potoków – czy to wbudowany, czy dostępny jako wtyczka (plug-in) – często pozwalają także na definiowanie samego potoku za pomocą skryptu przechowywanego w tym samym repozytorium. Takie rozwiązanie sprawia, że procesem CI/CD mogą zarządzać sami programiści, a nie tylko zespół odpowiedzialny za wdrożenie. Skrypt taki podlega też wersjonowaniu oraz procesom zwykłym dla kodu oprogramowania, takim jak Code Review, co zapewnia wgląd do historii modyfikacji i sprzyja utrzymaniu jego jakości, a co za tym idzie samego procesu integracji i wdrożenia.

W nowoczesnych architekturach systemów, bazujących na mikroserwisach, automatyzacja procesów wdrożenia staje się niezbędna, bo niezbędna jest modyfikacja wielu zależnych od siebie serwisów (https://softwareskill.pl/monitoring-mikrouslug). Nowoczesne narzędzia wspierające procesy CI i CD pomagają również we wdrożeniach tego typu, dając możliwość integracji m.in. z narzędziami takimi, jak wspomniany Kubernetes, a także monitorowania całego procesu czy przeprowadzania wdrożeń kanarkowych.

Każdy etap potoku składa się z jednego lub kilku zadań, z których część może być wykonywana równolegle na różnych agentach, przy czym zadania mogą też być od siebie zależne. Stosowanie potoków to krańcowy etap automatyzacji procesu dostarczania oprogramowania, ale zarówno one, jak i inne rozwiązania CI/CD zdejmują z zespołów DevOps odpowiedzialność za czasochłonne czynności, które mogą zostać wykonane automatycznie. Jednocześnie taka forma automatyzacji nie jest nakierowana na wyeliminowanie pracy ludzi na rzecz komputerów, co mogłoby być jej negatywną konsekwencją, a jedynie sprawia, że zespoły są w stanie w większym stopniu skupić się na samym na ulepszaniu tworzonego oprogramowania i interakcji z klientami i użytkownikami. Eliminują one także wiele błędów wynikających z konieczności manualnej konfiguracji czy manualnego wykonywania testów.

Upowszechnienie architektury mikroserwisowej i konteneryzacji przy powstawaniu nowych i rozbudowie istniejących systemów wymaga zapewnienia odpowiedniego poziomu monitorowania. Czynności takie, jak przeglądanie logów, debugowanie i analiza błędów czy sprawdzanie kondycji systemu (średni czas odpowiedzi, liczba pojawiających się błędów itp.), stają się znacznie bardziej skomplikowanymi procesami. Z pomocą przychodzą nam nowe narzędzia i wzorce postępowania, które świetnie sprawdzają się podczas pracy z systemami wdrożonymi w chmurze.

Agregacja logów

W przypadku systemu uruchomionego jako jeden proces, na jednej maszynie, przeglądanie logów nie stanowi

większego problemu. Problemy zaczynają się pojawiać, gdy aplikacja zostaje wdrożona w postaci wielu współpracujących ze sobą serwisów. Przeanalizowanie logów w takim systemie wymagałoby utworzenia połączenia do każdego serwisu osobno, co – przy rosnącej ich liczbie – staje się uciążliwe. Wraz z wykorzystaniem kontenerów dochodzi również problem ich efemeryczności.

Z pomocą przychodzą systemy agregujące logi. Dostarczają one rozwiązania służące do importowania logów z poszczególnych aplikacji do centralnego systemu. Taka synchronizacja może odbywać się na różne sposoby. Trzy najbardziej popularne to:

- wysyłanie logów bezpośrednio z serwisu do centralnego systemu,
- agent monitorujący zainstalowany na tej samej maszynie i uruchomiony obok aplikacji jako niezależny proces,
- tzw. *sidecar*, czyli dodatkowy kontener obok kontenera z aplikacją.

Wybrane rozwiązanie nie powinno istotnie wpływać na wydajność samej aplikacji. Dodatkową funkcjonalnością systemów agregujących jest możliwość wizualizacji logów, bardzo pomocna podczas analizy działania systemu – ponieważ logi są już odpowiednio sparsowane, co ułatwia skupienie się na ich treści. Istnieje również możliwość tworzenia zapytań filtrujących i przeszukujących logi po zadanych kryteriach, co z kolei może zostać wykorzystane do tworzenia alertów na ich podstawie.

Monitorowanie stanu aplikacji

Podobnie jak w przypadku przeglądania logów, stopień złożoności systemu utrudnia analizę błędów w działaniu aplikacji i bez odpowiednich narzędzi trudno byłoby zdiagnozować problemy. Pomocny staje się tutaj wzorzec *distributed tracing*, który daje możliwość prześledzenia requestów od momentu ich wejścia do systemu aż do uzyskania odpowiedzi (https://docs.logz.io/user-guide/distributed-tracing/what-is-tracing/).

Odbywa się to przez nadanie żądaniom unikalnego identyfikatora (tzw. *trace ID*), który później przekazywany jest we wszystkich wywołaniach, jakie następują po drodze. Następnie taka wartość dodawana jest do każdego wpisu w logach, co pozwala osadzić każdy z nich w odpowiednim kontekście. Dodatkowo każdy *trace* składa się z wielu *spanów*, czyli pojedynczych operacji wykonywanych w danej ścieżce wykonania. Mając odpowiednio skonfigurowany centralny system logowania oraz zaimplementowany wzorzec *distributed tracing* możemy debugować problemy w środowisku rozproszonym. Możliwa jest również wizualizacja przepływów wraz z czasem ich wykonania.

Równie ważnym elementem monitorowania aplikacji są metryki, które pozwalają ocenić ogólny stan systemu lub pojedynczych serwisów. Metryki możemy podzielić na biznesowe (np. liczba wygenerowanych faktur, zrealizowanych płatności w jednostce czasu) i techniczne (np. czas odpowiedzi poszczególnych zapytań, liczba żądań, zużycie CPU itp.). Jednostki stosowane przy definiowaniu metryk mogą być oczywiście różne, ale bardzo często wykorzystuje się percentyle, które najlepiej obrazują daną wartość. Gromadzenie statystyk bazuje często na wykorzystaniu logów, jakie generuje aplikacja. Dostępne są również specjalne narzędzia, które pozwalają na publikowanie metryk z systemu w ustandaryzowany sposób oraz ich wizualizowanie. Metryki są też dobrym punktem wyjściowym do zdefiniowania bazującego na nich systemu powiadamiania specjalistów. Metryki mogą zostać bardziej sformalizowane w postaci SLA (*Service Level Agreement* – umowa o poziomie świadczenia usług), co powoduje, że stają się one obiektem zainteresowania nie tylko inżynierów, lecz także ludzi biznesu.

Wyzwania organizacyjne

Organizacja pracy w zespołach DevOps – złożonych ze specjalistów tworzących oprogramowanie i zarządzających nim, działających w trybie *Continuous Development/Continuous Delivery* dzięki „skróceniu pętli sprzężenia zwrotnego" – okazała się bardzo skutecznym sposobem zarówno szybkiego dostarczania działającego oprogramowania, jak i szybkiego modyfikowania lub poprawiania go na podstawie bieżących analiz działania programów. Choć w takiej organizacji pracy pojawiły się nowe role zawodowe (np. DevOps developer) oraz odpowiednie narzędzia ją wspierające, to organizacja DevOps nie spowodowała radykalnych zmian w inżynierii oprogramowania. Zdecydowana większość narzędzi oraz metod tworzenia i testowania programów czy mikroserwisów powstała w zasadzie niezależnie od pojawienia się w firmach zespołów DevOps.

Z daleko większymi zmianami wiąże się organizacja pracy typu BizDevOps, którą widać w firmach i instytucjach nieinformatycznych i która polega na bezpośredniej współpracy działów biznesowych z własnymi działami IT firmy, wspólnego działania pracowników biznesowych ze specjalistami z zewnętrznych firm IT czy współpracy w ramach tzw. body leasingu. Procesem toczącym się równolegle jest rozwój narzędzi nisko- i zerokodowych, umożliwiających tworzenie rozwiązań informatycznych bez umiejętności kodowania w jakimkolwiek języku programowania. Z jednej strony narzędzia Low-Code/No-Code rozszerzają możliwości pracy w układzie BizDevOps, a z drugiej strony – organizacja pracy w zespołach BizDevOps stymuluje czy wręcz wymusza rozwijanie takich narzędzi.

Organizacja pracy w modelu BizDevOps z wykorzystaniem zaawansowanych narzędzi nisko- i zerokodowych stawia przed wprowadzającymi je firmami i instytucjami wymagania przede wszystkim w trzech obszarach:

- **kompetencji pracowników-nieinformatyków i specjalistów IT,**
- **organizacji procesów biznesowych,**
- **organizacji współpracy pracowników z rozwiązaniami zautomatyzowanymi.**

W obszarze kompetencji pracowników działów nieinformatycznych wymagane są umiejętności posługiwania się nowymi narzędziami w taki sposób, by móc z ich użyciem odpowiednio informatyzować dobrze znane tym pracownikom procesy biznesowe i procedury administracyjne. Specjaliści IT muszą umieć tworzyć narzędzia Low-Code/No-Code, w których coraz szerzej stosowane są rozwiązania sztucznej inteligencji (AI) i uczenia maszynowego (ML). Pojawiają się nowe role zawodowe, np. „pasterz robotów" – specjalista, który potrafi odpowiednio prowadzić proces nauczania robotów programowych. Uczenie robotów wymaga także analizy i odpowiedniego doboru danych treningowych, aby uniknąć „nauczenia" robota nieprawidłowych decyzji i działań, wynikających ze złego doboru danych treningowych i testowych. Natomiast ci informatycy, którzy pracują w zespołach BizDevOps, muszą coraz lepiej rozumieć procesy biznesowe i organizacyjne, żeby pomagać nieinformatykom w korzystaniu z narzędzi Low-Code/No-Code, a także wspólnie definiować założenia i wymagania do tworzenia takich narzędzi.

W obszarze organizacji procesów są już doświadczenia robotyzacji procesów biznesowych – np. w bankach, u operatorów telekomunikacyjnych, dostawców różnych usług masowych czy wielkich platform sprzedaży – a więc w firmach i organizacjach, w których najszerzej wprowadzane są roboty programowe, a następnie kompleksowe rozwiązania nazywane hiperautomatyzacją. Wynika z nich, że mimo rozwoju zastosowań AI/ML warunkiem sukcesu, a często wręcz możliwości rozpoczęcia automatyzacji, jest uporządkowanie i sformalizowanie procesów i procedur. Prawidłowa organizacja i formalizacja procesów wymaga odpowiednich kompetencji nie tylko pracowników na szczeblu bezpośrednio produkcyjnym, lecz także menedżerskim i zarządczym.

Z punktu widzenia inżynierii oprogramowania robotyzacja i hiperautomatyzacja niesie ze sobą konieczność odpowiedzi na kilka pytań dotyczących odpowiedzialności za jakość rozwiązań informatycznych tworzonych przy wykorzystaniu AI/ML. Do rozstrzygnięcia są np. takie kwestie, jak to, kto odpowiada za wydajność, spójność i bezpieczeństwo stosowania rozwiązań tworzonych narzędziami Low-Code/No-Code przez nieinformatyków.

Obszar współpracy pracowników z robotami do tej pory koncentrował się na bezpieczeństwie ludzi współpracujących z robotami stosowanymi w produkcji przemysłowej (np. roboty spawalnicze, zrobotyzowane gniazda maszynowe itp.). Pojawienie się coraz bardziej zaawansowanych robotów programowych zwróciło uwagę na zagadnienia tzw. kobotyzacji – współpracy ludzi z rozwiązaniami korzystającymi ze sztucznej inteligencji. Jest to obszar, który wymaga nowych kompetencji specjalistów HR w firmach i instytucjach m.in. z uwagi na jego aspekty psychologiczne i oddziaływania społeczne. To zagadnienie wykracza jednak daleko poza dziedzinę inżynierii oprogramowania i zasługuje na oddzielne omówienie.