O'REILLY®

Kubernetes
– rozwiązania chmurowe w świecie DevOps

Tworzenie, wdrażanie i skalowanie nowoczesnych aplikacji chmurowych

John Arundel
Justin Domingus

Helion

Tytuł oryginału: Cloud Native DevOps with Kubernetes: Building, Deploying,
and Scaling Modern Applications in the Cloud

Tłumaczenie: Łukasz Wójcicki

ISBN: 978-83-283-6927-6

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/kubdev
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 1.

Rewolucja chmurowa

Nie wiemy, kiedy świat zaczął istnieć, ponieważ krąży po okręgu, a okrąg nie ma początku.

— Alan Watts

Trwa rewolucja. Właściwie trzy rewolucje.

Pierwszą rewolucją jest utworzenie chmury — wyjaśnimy Ci, czym jest. Drugą — wprowadzenie metodyki *DevOps* — dowiesz się, jakie obejmuje operacje. Trzecią rewolucją jest powstanie kontenerów. Te trzy fale zmian razem tworzą nowy świat oprogramowania zwany *cloud native*. System operacyjny tego świata to *Kubernetes*.

W tym rozdziale krótko opiszemy historię i znaczenie tych zmian. Sprawdzimy także, w jaki sposób wpływają one na sposób wdrażania i obsługi oprogramowania. Wyjaśnimy także, co oznacza pojęcie *cloud native* i jakich zmian możesz spodziewać się w środowisku, w którym aktualnie pracujesz — o ile zajmujesz się rozwojem oraz wdrażaniem oprogramowania, bezpieczeństwem czy też administracją sieci.

Kiedy powiążemy powyższe trzy technologie, odkryjemy, że przyszłość rozwoju oprogramowania leży w rozwiązaniach chmurowych, konteneryzacji, systemach rozproszonych dynamicznie zarządzanych za pomocą automatyzacji na platformie *Kubernetes* (lub innej podobnej). Dalej w książce zapoznamy Cię ze sposobami obsługi tych aplikacji — rozwiązaniami chmurowymi opartymi na metodyce *DevOps*.

Jeżeli już znasz pojęcia podstawowe, możesz rozpocząć zabawę z Kubernetes, zaczynając od rozdziału 2. Jeśli nie, usiądź wygodnie z kubkiem swojego ulubionego napoju. Zaczynamy.

Tworzenie chmury

Początkowo (czyli w latach 60. ubiegłego wieku) komputery wypełniały szafy w olbrzymich, klimatyzowanych centrach danych — bez bezpośredniego dostępu użytkowników. Programiści pracowali na nich zdalnie. W taki sam sposób do takiej infrastruktury obliczeniowej mogłoby się dostać wiele tysięcy użytkowników — za zużyty czas procesora lub inne zasoby otrzymaliby rachunek.

Dla wielu firm lub organizacji zakup i utrzymanie własnego sprzętu komputerowego nie byłyby opłacalne. Dlatego pojawił się model biznesowy, w którym użytkownicy dzielą się mocą obliczeniową zdalnych maszyn będących własnością trzeciej strony.

Jeśli przypomina to dzisiejszą sytuację, nie jest to przypadek. Słowo rewolucja powstało od słowa „obrót", a informatyka w pewnym sensie powróciła do miejsca, w którym się rozpoczęła. Komputery na przestrzeni lat stały się o wiele mocniejsze — dzisiejszy zegarek Apple stanowi odpowiednik około trzech komputerów mainframe. Jak pokazano na rysunku 1.1, współużytkowany dostęp do zasobów obliczeniowych typu pay-per-use to bardzo stary pomysł. Obecnie taki sposób dostępu nazwany został *rozwiązaniem chmurowym*. Rewolucja, która rozpoczęła się wraz ze współdzieleniem komputerów mainframe, zatoczyło koło.

Rysunek 1.1. *Wczesny komputer chmurowy: IBM System/360 Model 91 w NASA Goddard Space Flight Center*

Czas kupowania

Główną ideą rozwiązań chmurowych jest to, że zamiast zakupu *komputera* kupujesz *moc obliczeniową*. Oznacza to, że nie lokujesz sporej ilości kapitału w fizyczny zakup dużej ilości sprzętu komputerowego, który jest trudny do skalowania, łatwo się psuje oraz szybko starzeje. Zamiast tego płacisz tylko za czas pracy na komputerze, który jest własnością kogoś innego. Ten ktoś, a nie Ty, musi się zająć skalowaniem, konserwacją oraz aktualizacją tego sprzętu. W czasie epoki maszyn typu bare-metal — lub jeśli wolisz „epoki żelaza" — moc obliczeniowa była wydatkiem kapitałowym. Obecnie jest wydatkiem operacyjnym.

Chmura to nie tylko zdalna, wynajęta moc obliczeniowa. To także rozproszone systemy operacyjne. Możesz kupić tylko moc obliczeniową (tak jak w np. instancji Google Compute lub usłudze AWS Lambda) i użyć jej do uruchomienia własnego oprogramowania. Możesz także skorzystać z *usług chmurowych* (ang. *cloud services*) — czyli użyć czyjegoś oprogramowania. Jeśli np. korzystasz z oprogramowania PagerDuty do monitorowania i ostrzegania (w przypadku wystąpienia awarii systemu), używasz usługi w chmurze (czasami nazywanej usługą *oprogramowanie jako usługa*, w skrócie SaaS).

Infrastruktura jako usługa

Kupując infrastrukturę chmurową dla swoich serwisów, naprawdę kupujesz usługę zwaną *infrastruktura jako usługa* (IaaS). Na jej zakup nie musisz korzystać z wydatków kapitałowych — nie musisz jej budować czy też ulepszać. To tylko koszt, taki jak elektryczność lub woda. Rozwiązania chmurowe to rewolucja w relacjach pomiędzy prowadzeniem interesu a infrastrukturą IT.

Outsourcing sprzętu to tylko część historii; chmura pozwala również na outsourcing oprogramowania, którego sam nie stworzysz; są to systemy operacyjne, bazy danych, oprogramowanie do klastrowania, replikacji, zarządzania siecią, monitorowania, zapewniające dostępność, obróbkę strumieni oraz wszystkie niezliczone warstwy oprogramowania i konfiguracji, które wypełniają lukę pomiędzy Twoim kodem a procesorem. Usługi zarządzane (ang. *managed services*) mogą obsłużyć prawie wszystkie te banalne ciężary (ang. *undifferentiated heavy lifting*) za Ciebie (o zaletach zarządzanych usług dowiesz się więcej w rozdziale 3.).

Rewolucja w chmurze wywołała także kolejną rewolucję w ludziach, którzy z niej korzystają; stał się nią ruch w kierunku DevOps.

Początki DevOps

Przed powstaniem metodyki DevOps tworzenie i obsługa oprogramowania były zasadniczo dwoma oddzielnymi zadaniami, obsługiwanymi przez dwie różne grupy ludzi. *Programiści* po napisaniu przekazywali oprogramowanie personelowi *operacyjnemu*, który je uruchamiał i walczył z problemami w środowisku *produkcyjnym* (tzn. takim, które obsługuje prawdziwych użytkowników, zamiast po prostu działać w warunkach testowych). Podobnie jak komputery, które potrzebują własnej przestrzeni na podłodze, ta separacja ma swoje korzenie w połowie ubiegłego wieku. Tworzenie oprogramowania było bardzo specjalistycznym zajęciem — tak jak zarządzanie infrastrukturą informatyczną. Oba obszary nakładały się na siebie tylko w małym stopniu.

W rzeczywistości dwa działy miały zupełnie inne cele, które często kolidowały (patrz rysunek 1.2). Programiści zwykle koncentrują się na szybkim dostarczeniu nowych funkcjonalności, podczas gdy zespoły operacyjne dbają o to, aby dostarczone usługi były stabilne w dłuższym okresie czasu.

Kiedy na horyzoncie pojawiły się rozwiązania chmurowe, wszystko się zmieniło. Systemy rozproszone są złożone, a internet jest bardzo duży. Technikalia związane z obsługą systemu — przywracanie po awarii, obsługa timeoutów, płynna aktualizacja wersji — nie są tak po prostu oddzielone od tworzenia, architektury oraz implementacji systemu.

Rysunek 1.2. *Wśród dwóch zespołów może dochodzić do konfliktu interesów (autor Dave Roth)*

Co więcej, „system" nie jest już tylko Twoim oprogramowaniem: obejmuje oprogramowanie wewnętrzne, usługi chmurowe, zasoby sieciowe, load balancery, monitorowanie, sieci dystrybucji, firewalle, DNS itd. Wszystkie te rzeczy są ściśle powiązane i współzależne. Ludzie, którzy piszą oprogramowanie, muszą zrozumieć, w jaki sposób odnosi się do reszty systemu, a osoby obsługujące system muszą zrozumieć, jak działa oprogramowanie.

Ruch DevOps ma na celu połączenie tych dwóch grup. Dzięki współpracy można podzielić odpowiedzialność za systemy, poprawić niezawodność działania oraz zwiększyć skalowalność obu systemów i zespołów ludzi, którzy je zbudowali.

Nikt nie rozumie DevOps

DevOps był w pewnym okresie kontrowersyjnym pomysłem zarówno dla ludzi, którzy mówili, że to nic więcej, tylko nowoczesny opis już istniejących dobrych praktyk tworzenia oprogramowania, jak i dla tych, którzy odrzucali potrzebę ściślejszej współpracy między tworzeniem oprogramowania a działaniami operacyjnymi.

Istnieje również powszechne nieporozumienie dotyczące tego, czym właściwie jest DevOps. Czy to rodzaj pracy? Czy zespół? Czy metodologia? Czy zestaw umiejętności? Wpływowy autor opisujący DevOps John Willis zidentyfikował cztery kluczowe filary DevOps, takie jak kultura, automatyzacja,

metryki i współdzielenie (CAMS — ang. *culture*, *automation*, *measurment*, *sharing*). Inaczej przedstawił to Brian Dawson. Stworzył on tzw. trójcę DevOps: ludzie i kultura, proces i praktyka oraz narzędzia i technologia.

Niektórzy uważają, że istnienie chmur i kontenerów oznacza, że nie potrzebujemy już DevOps — taki punkt widzenia nazywany jest czasem *NoOps*. Oznacza to, że skoro wszystkie operacje IT są wynajmowane u usługodawcy w chmurze lub za pomocą usług trzeciej strony, to firmy nie potrzebują personelu operacyjnego w pełnym wymiarze godzin.

Założenia NoOps opierają się na niezrozumieniu tego, co faktycznie oferuje DevOps.

> *Dzięki DevOps większość tradycyjnych operacji informatycznych ma miejsce przed powstaniem kodu produkcyjnego. Każde wydanie nowej wersji oprogramowania obejmuje monitorowanie, rejestrowanie i testowanie A / B. Potoki CI/CD automatycznie uruchamiają testy jednostkowe, testy bezpieczeństwa oraz kotrole polityk przy każdym zatwierdzeniu każdej zmiany oprogramowania (ang.* commit*). Wdrożenia są automatyczne. Kontrole, zadania i wymagania niefunkcjonalne są teraz implementowane przed każdym wydaniem oprogramowania, co pozwala uniknąć niebezpiecznych następstw wystąpienia krytycznej awarii.*
>
> — Jordan Bach (AppDynamics (*https://blog.appdynamics.com/engineering/is-noops-the-end-of-devops-think-again/*))

Aby zrozumieć DevOps, trzeba wiedzieć, że dotyczy ono przede wszystkim kwestii organizacyjnych, ludzkich, a nie technicznych. Stwierdzenie to nawiązuje do jednego z praw o nazwie *Second Law of Consulting*, którego autorem jest Gerald Weinberg.

> *Mimo iż na początku może wyglądać inaczej, to zawsze jest problem ludzki.*
>
> — Gerald M. Weinberg, Secrets of Consulting

Przewaga biznesowa

Z biznesowego punktu widzenia DevOps został opisany jako „poprawa jakości oprogramowania poprzez przyśpieszenie cyklów wydawniczych z użyciem chmurowych rozwiązań oraz wprowadzenie nowych korzyści dla oprogramowania, które jest obecnie w produkcji" (The Register (*https://www.theregister.co.uk/2018/03/06/what_does_devops_do_to_decades_old_planning_processes_and_assumptions*)).

Przyjęcie DevOps wymaga głębokiej transformacji kulturowej dla firm. Muszą one rozpocząć zmiany na poziomie wykonawczym, strategicznym i stopniowo rozprzestrzeniać się w każdej części organizacji. Szybkość, zwinność, współpraca, automatyzacja i jakość oprogramowania są kluczowymi czynnikami DevOps — dla wielu firm oznacza to znaczną zmianę sposobu myślenia.

Jednak DevOps działa, a badania regularnie wykazują, że firmy, które przyjmują zasady DevOps, szybciej wypuszczają lepsze oprogramowanie, lepiej i szybciej reagują na awarie, są bardziej zwinne na rynku oraz znacznie poprawiają jakość swoich produktów.

DevOps nie jest chwilową modą; jest to raczej sposób, w jaki odnoszące sukcesy organizacje uprzemysławiają dostarczanie wysokiej jakości oprogramowania; będzie to stanowiło także podstawę przez wiele kolejnych lat.

— Brian Dawson (Cloudbees), Computer Business Review (*https://www.cbronline.com/enterprise-it/applications/devops-fad-stay*)

Infrastruktura w postaci kodu

Dawno, dawno temu programiści tworzyli oprogramowanie, a zespoły operacyjne zajmowały się sprzętem oraz systemami operacyjnymi.

Obecnie, gdy sprzęt znajduje się w chmurze, wszystko w pewnym sensie jest oprogramowaniem. Ruch DevOps przenosi tworzenie programowania do operacji; są to narzędzia i przepływy pracy do szybkiego, zwinnego, wspólnego budowania złożonych systemów. Z DevOps nierozerwalnie związane jest pojęcie *infrastruktury jako kodu*.

Fizyczna infrastruktura, w postaci komputerów, switchów, okablowania może zostać zastąpiona, za sprawą oprogramowania, przez infrastrukturę chmurową. Obowiązki inżynierów operacyjnych ulegają zmianie. Wcześniej musieli ręcznie dbać o upgrade sprzętu, a po zmianie są ludźmi tworzącymi oprogramowanie, które automatyzuje pracę w środowisku chmurowym.

Transformacja nie przebiega tylko w jednym kierunku. Programiści uczą się od zespołów operacyjnych, jak przewidywać awarie związane z rozproszonymi systemami opartymi na chmurze, jak złagodzić ich konsekwencje i jak zaprojektować bezpieczne i zgodne wstecz oprogramowanie.

Wspólna nauka

Zarówno zespoły programistów, jak i zespoły operacyjne uczą się, jak pracować razem. Uczą się, jak projektować i budować systemy, jak monitorować systemy produkcyjne oraz w jaki sposób wykorzystać uzyskane informacje w celu usprawnienia działania. Co ważniejsze, uczą się poprawiać jakość obsługi swoich użytkowników oraz podnoszą wartość firmy.

Ogromne możliwości środowiska chmurowego i oparty na kodzie charakter metodyki DevOps sprawiły, że wszelkie operacje stały się problemem oprogramowania. Jednocześnie oprogramowanie stało się problemem operacyjnym, zatem nasuwają się następujące pytania.

- W jaki sposób wdrażać i aktualizować oprogramowanie w dużych, różnorodnych sieciach, z różnymi architekturami serwerów i systemów operacyjnych?
- W jaki sposób wdrażać oprogramowanie w środowiskach rozproszonych w niezawodny i powtarzalny sposób, używając w dużej mierze standardowych komponentów?

Czas na trzecią rewolucję, czyli kontener.

Nadejście kontenerów

Aby wdrożyć oprogramowanie, potrzebujesz nie tylko samego oprogramowania, ale także składników wymaganych przez to oprogramowanie. Są to biblioteki, interpretery, pakiety, kompilatory, rozszerzenia itd.

Potrzebujesz także *konfiguracji*. Ustawienia, szczegółowe informacje dotyczące witryny, klucze licencyjne, hasła do bazy danych, wszystko, co zmienia oprogramowanie w użyteczną usługę.

Stan aktualny

Wcześniejsze próby rozwiązania tego problemu obejmują korzystanie z systemów *zarządzania konfiguracją*, takich jak Puppet lub Ansible. Narzędzia te dostarczają kod umożliwiający instalację, uruchomienie, konfigurację oraz aktualizację oprogramowania.

Niektóre języki zapewniają własny mechanizm archiwizacji, np. pliki JAR Javy, eggi Pythona lub gemy Ruby. Są one jednak specyficzne dla języka i nie rozwiązują całkowicie problemu zależności: aby np. uruchomić plik JAR, nadal potrzebujesz zainstalowanego środowiska wykonawczego Java.

Innym rozwiązaniem jest pakiet *omnibus*, który — jak sugeruje nazwa — próbuje wcisnąć wszystko, czego aplikacja potrzebuje, do jednego pliku. Taki pakiet zawiera oprogramowanie, jego konfigurację, zależne komponenty, ich konfigurację, ich zależności itd. (Przykładowo pakiet omnibus Java zawierałby środowisko wykonawcze Java, a także wszystkie pliki JAR aplikacji).

Niektórzy producenci poszli nawet o krok dalej i umieścili cały system komputerowy jako *obraz maszyny wirtualnej*. Obrazy te są jednak duże, czasochłonne w budowie i utrzymaniu, trudne we wdrażaniu oraz ogromnie nieefektywne i zasobożerne.

Z operacyjnego punktu widzenia nie tylko musisz zarządzać różnego rodzaju pakietami, ale także zarządzać grupą serwerów.

Serwery muszą być udostępniane, łączone w sieć, wdrażane, konfigurowane, aktualizowane za pomocą poprawek bezpieczeństwa, monitorowane, zarządzane itd.

Wszystkie powyższe działania wymagają znacznej ilości czasu, umiejętności i wysiłku. Czy nie ma lepszego sposobu?

Myślenie pudełkowe

Aby rozwiązać te problemy, przemysł technologiczny zapożyczył pomysł z branży morskiej, czyli kontener. W latach 50. ubiegłego wieku kierowca ciężarówki o nazwisku Malcolm McLean (*https://hbs.me/2Q0QCzb*) zaproponował, aby zamiast żmudnego rozładowywania towarów z ciężarówek ładować na statek od razu całą zawartość ciężarówki.

Przyczepa samochodowa to zasadniczo duża metalowa skrzynia na kołach. Jeśli możesz oddzielić pudło — kontener — od kół i podwozia używanego do jego transportu, to uzyskujesz coś, co jest bardzo łatwe do podniesienia, załadowania, ułożenia i rozładowania. Może zostać bezpośrednio przeniesione na statek lub inną ciężarówkę (patrz rysunek 1.3).

Rysunek 1.3. *Standaryzowane pojemniki znacznie obniżają koszty wysyłki towarów masowych (zdjęcie z serwisu Pixabay (https://www.pexels.com/@pixabay), licencja Creative Commons 2.0)*

Sea-Land, firma kurierska McLeana odniosła duży sukces. Wykorzystała ten system do znacznie tańszej wysyłki towarów w kontenerach (*https://www.freightos.com/the-history-of-the-shipping-container*). Obecnie każdego roku wysyłane są setki milionów kontenerów, które przewożą towary o wartości trylionów dolarów.

Umieszczanie oprogramowania w kontenerach

Kontener oprogramowania to dokładnie ten sam pomysł: standardowy format pakowania i dystrybucji, który jest ogólny i rozpowszechniony. Umożliwia znacznie większą nośność, niższe koszty, skalowanie i łatwość obsługi. Kontener zawiera wszystko, czego aplikacja potrzebuje do uruchomienia. Zapisany jako plik *obrazu* może być wykonany przez środowisko *wykonawcze kontenera*.

Jaka jest różnica w stosunku do obrazu maszyny wirtualnej? On także zawiera wszystko, co aplikacja musi uruchomić — ale o wiele więcej. Typowy obraz maszyny wirtualnej to około 1 GiB[1]. Z drugiej strony, dobrze zaprojektowany obraz kontenera może być sto razy mniejszy.

Ponieważ maszyna wirtualna zawiera wiele aplikacji, bibliotek i innych rzeczy, których nasze oprogramowanie nigdy nie będzie używać, większość miejsca jest marnowana. Przesyłanie obrazów maszyn wirtualnych w sieci jest znacznie wolniejsze niż zoptymalizowanych kontenerów.

[1] Gibibajt (GiB) to jednostka danych Międzynarodowej Komisji Elektrotechnicznej (IEC), zdefiniowana jako 1 024 mebibajtów (MiB). W tej książce będziemy używać jednostek IEC (GiB, MiB, KiB), aby uniknąć niejasności.

Co gorsza, maszyny wirtualne są tylko *wirtualne*: fizyczny CPU *emuluje* procesor, na którym działa maszyna wirtualna. Warstwa wirtualizacji ma dramatyczny, negatywny wpływ na wydajność (*https://www.stratoscale.com/blog/container/running-containers-on-bare-metal/*); w testach wirtualizacje działają o około 30% wolniej od równoważnych kontenerów.

Dla porównania, kontenery działają pod kontrolą rzeczywistego CPU, bez narzutów związanych z wirtualizacją, tak jak robią to zwykłe pliki binarne.

A ponieważ kontenery zawierają tylko potrzebne pliki, są znacznie mniejsze niż obrazy maszyn wirtualnych. Użyto w nich również sprytnej techniki adresowalnych warstw systemu plików, które mogą być współużytkowane i ponownie wykorzystywane między kontenerami.

Jeśli np. masz dwa kontenery i oba zawierają ten sam obraz Debian Linux, jest on pobierany tylko raz, a każdy kontener może po prostu się do niego odwoływać.

Środowisko wykonawcze kontenera zgromadzi wszystkie niezbędne warstwy i pobierze warstwę tylko wtedy, gdy nie jest ona lokalnie buforowana. W ten sposób bardzo efektywnie wykorzystywane jest zarówno miejsce na dysku, jak i przepustowość sieci.

Aplikacje Plug and Play

Kontener jest nie tylko jednostką wdrożenia oraz jednostką pakietu, jest to także jednostka ponownego *wykorzystania* (ten sam obraz kontenera może być wykorzystany jako składnik wielu różnych usług), jednostka skalowania i jednostka *alokacji zasobów* (kontener może działać wszędzie tam, gdzie dostępne są wystarczające zasoby na jego specyficzne potrzeby).

Programiści nie muszą się już martwić problemami związanymi z utrzymywaniem różnych wersji oprogramowania, tak aby działały w różnych dystrybucjach Linuksa, w różnych wersjach bibliotek, w różnych wersjach języka programowania itd. Praca z kontenerami zależy jedynie od jądra systemu operacyjnego (np. Linuksa).

Po prostu dodaj swoją aplikację do obrazu kontenera, a będzie ona działać na dowolnej platformie, która obsługuje standardowy format kontenera i ma kompatybilne jądro.

Brendan Burns i David Oppenheimer, programiści Kubernetes, opisali to w swoim artykule *Design Patterns for Container-based Distributed Systems* (*https://www.usenix.org/node/196347*).

> *Dzięki hermetycznemu zamknięciu, zadbaniu o zależności i wprowadzeniu atomowego sygnału wdrażania („sukces” / „błąd”) [kontenery] znacznie poprawiają dotychczasowy stan wdrażania oprogramowania w centrum danych lub chmurze. Jednak kontenery mogą być potencjalnie czymś więcej niż tylko lepszym narzędziem do wdrażania — wierzymy, że ich działanie może być analogiczne do obiektów w obiektowych systemach oprogramowania i jako takie umożliwi opracowanie wzorców projektowania systemów rozproszonych.*

Uruchomienie orkiestratora

Zespoły operacyjne również doszły do wniosku, że użycie kontenerów spowodowało, iż ich praca została uproszczona. Zamiast utrzymywać rozległą posiadłość maszyn różnego rodzaju, architektur i systemów operacyjnych, wystarczy uruchomić *orkiestrator kontenerów* (ang. *container orchestrator*),

czyli oprogramowanie zaprojektowane do łączenia wielu różnych maszyn w jeden klaster. Są to połączone jednostki obliczeniowe, postrzegane przez użytkownika jako pojedynczy bardzo wydajny komputer (na którym mogą działać kontenery).

Terminy *orkiestracja* (ang. *orchestration*) i *planowanie* (ang. *scheduling*) są często używane jako synonimy. Ściśle mówiąc, *orkiestracja* w tym kontekście oznacza koordynację i sekwencjonowanie różnych działań na rzecz wspólnego celu (podobnie do pracy muzyków w orkiestrze). *Planowanie* oznacza zarządzanie dostępnymi zasobami i przypisywanie prac tam, gdzie można je najbardziej wydajnie uruchomić. (Nie należy mylić z planowaniem w sensie zaplanowanych zadań, które są wykonywane o ustalonych porach).

Trzecim ważnym działaniem jest *zarządzanie klastrami* (ang. *cluster management*), czyli łączenie wielu serwerów fizycznych lub wirtualnych w zunifikowaną, niezawodną, odporną na awarie grupę.

Termin „orkiestrator kontenerów" zwykle odnosi się do pojedynczej usługi, która zajmuje się planowaniem i koordynacją klastra oraz zarządzaniem nim.

Konteneryzacja (używanie kontenerów jako standardowej metody wdrażania i uruchamiania oprogramowania) zapewniała oczywiste korzyści, a standardowy format kontenerów umożliwiał wszelkiego rodzaju korzyści. Jednak wciąż jeden problem przeszkadzał w upowszechnieniu kontenerów; był to brak standardowego systemu orkiestracji kontenerów.

Tak długo, jak kilka różnych narzędzi do planowania i orkiestracji kontenerów konkurowało na rynku, firmy niechętnie decydowały się na wybór konkretnych technologii. Jednak to wszystko miało się zmienić.

Kubernetes

Google skorzystał z kontenerów na dużą skalę znacznie wcześniej niż ktokolwiek inny. Prawie wszystkie usługi Google działają w kontenerach: Gmail, wyszukiwarka Google, Mapy Google, Google App Engine itd. Ponieważ w tym czasie nie istniał żaden odpowiedni system orkiestracji kontenerów, Google został zmuszony do jego wynalezienia.

Od Borga do Kubernetes

Aby rozwiązać problem związany z uruchomieniem dużej liczby usług w skali globalnej na milionach serwerów, Google opracował prywatny, wewnętrzny system orkiestracji kontenerów o nazwie Borg (*https://pdos.csail.mit.edu/6.824/papers/borg.pdf*).

Borg zasadniczo jest scentralizowanym systemem zarządzania przydzielającym i planującym, które kontenery mają się uruchomić w danej puli serwerów. Bardzo potężny Borg jest ściśle powiązany z wewnętrznymi i zastrzeżonymi technologiami Google, trudnymi do rozszerzenia i niemożliwymi do publicznego udostępnienia.

W 2014 r. Google założył projekt o otwartym kodzie źródłowym o nazwie Kubernetes (od greckiego słowa κυβερν meaningτης, co oznacza sternik, pilot), który pozwala na wdrożenie orkiestratora kontenerów przez każdego, kto chce z niego skorzystać. Firma oparła się na wnioskach wyciągniętych z Borga i jego następcy Omegi (*https://research.google/pubs/pub41684/*).

Rozwój projektu Kubernetes był błyskawiczny. Mimo iż inne systemy orkiestracji kontenerów istniały przed Kubernetes, były to produkty komercyjne, zależne od producenta, co zawsze stanowiło barierę dla ich powszechnego wdrożenia. Wraz z pojawieniem się prawdziwie bezpłatnego i otwartego oprogramowania do tworzenia kontenerów wykorzystanie zarówno kontenerów, jak i Kubernetes rosło w niesamowitym tempie.

Pod koniec 2017 r. walki producentów zakończyły się, Kubernetes wygrał. Inne systemy są nadal w użyciu, jednak od teraz firmy, które chcą przenieść swoją infrastrukturę do kontenerów, stawiają tylko na jedną platformę: Kubernetes.

Co sprawia, że Kubernetes jest tak cenny?

Kelsey Hightower, rzecznik programistów Google, współautor *Kubernetes Up & Running* (O'Reilly) i legenda społeczności Kubernetes, przedstawia to w następujący sposób.

> *Kubernetes robi rzeczy, które zrobiłby najlepszy administrator systemu, takie jak automatyzacja, obsługa awarii, monitorowanie. Przejmuje wszystkie nauki ze społeczności DevOps i sprawia, że stają się domyślne.*
>
> — Kelsey Hightower

Wiele tradycyjnych zadań sysadminów, takich jak aktualizowanie serwerów, instalowanie poprawek bezpieczeństwa, konfigurowanie sieci i wykonywanie kopii zapasowych, nie stanowi problemu w środowisku chmurowym. Kubernetes może zautomatyzować te rzeczy, aby Twój zespół mógł skoncentrować się na wykonywaniu swojej podstawowej pracy.

Niektóre z tych funkcji, np. równoważenie obciążenia i automatyczne skalowanie, są wbudowane w rdzeń Kubernetes; inne są dostarczane przez dodatki, rozszerzenia i narzędzia innych firm korzystające z interfejsu API Kubernetes. Ekosystem Kubernetes jest duży i cały czas rośnie.

Kubernetes ułatwia wdrażanie

Z tych powodów załoga DevOps uwielbia Kubernetes. Są też pewne znaczące zalety dla programistów. Kubernetes znacznie skraca czas i wysiłek potrzebny do wdrożenia. W Kubernetes wdrożenia bez przestojów są powszechne, ponieważ Kubernetes domyślnie wykonuje aktualizacje ciągłe (uruchamia kontenery z nową wersją, następnie czeka, aż będą gotowe, i zamyka stare).

Kubernetes zapewnia również udogodnienia, które pomagają wprowadzić praktyki ciągłego wdrażania, takie jak *canary deployments*, czyli stopniowe wdrażanie aktualizacji na serwer, tak aby wcześnie wychwycić problemy (patrz „Wdrożenia kanarkowe" w rozdziale 13.). Inną powszechną praktyką są wdrożenia metodą blue-green: równoległe uruchamianie nowej wersji systemu i przełączanie do niego ruchu po jego pełnym uruchomieniu (patrz „Wdrożenia niebiesko-zielone" w rozdziale 13.).

Nagłe skoki zapotrzebowania na zasoby nie będą już obniżać poziomu usług, ponieważ Kubernetes obsługuje automatyczne skalowanie. Jeśli np. wykorzystanie procesora przez kontener osiągnie pewien poziom, Kubernetes może dodawać nowe repliki kontenera, dopóki wykorzystanie nie spadnie poniżej progu. Kiedy zapotrzebowanie spadnie, Kubernetes ponownie zmniejszy ilość replik, uwalniając pojemność klastra do uruchamiania innych prac.

Ponieważ Kubernetes ma wbudowaną redundancję i przełączanie awaryjne (ang. *failover*), aplikacja będzie bardziej niezawodna i odporna. Niektóre usługi zarządzane mogą nawet skalować sam klaster Kubernetes w górę i w dół w odpowiedzi na zapotrzebowanie, dzięki czemu nigdy nie płacisz za większy klaster niż potrzebujesz w danym momencie (patrz „Automatyczne skalowanie" w rozdziale 6.).

Firma pokocha Kubernetes, ponieważ obniża koszty infrastruktury i znacznie lepiej wykorzysta dany zestaw zasobów. Tradycyjne serwery, nawet serwery w chmurze, są zwykle bezczynne przez większość czasu. Nadwyżka mocy potrzebna do obsługi skoków zapotrzebowania na zasoby jest zasadniczo marnowana w normalnych warunkach.

Kubernetes wykorzystuje tę zmarnowaną pojemność i używa jej do uruchamiania zadań, a zatem maszyny są znacznie lepiej wykorzystywane — dodatkowo zyskujemy skalowanie, równoważenie obciążenia oraz obsługę awarii.

Chociaż niektóre z tych funkcji, takie jak automatyczne skalowanie, były dostępne przed Kubernetes, zawsze były powiązane z konkretnym dostawcą usług chmurowych lub usługą. Kubernetes działa w sposób niezależny od dostawcy (ang. *provider-agnostic*): po zdefiniowaniu zasobu do wykorzystania możesz go uruchomić w dowolnym klastrze Kubernetes, w sposób niezależny od dostawcy rozwiązań chmurowych.

To nie znaczy, że Kubernetes ogranicza Cię do najniższego wspólnego mianownika. Kubernetes mapuje Twoje zasoby do odpowiednich funkcji specyficznych dla dostawcy: np. dla usługi typu load-balance Kubernetes w Google Cloud zostanie utworzony jej odpowiednik, a w Amazon powstanie usługa AWS load balancer. Kubernetes wyodrębnia szczegóły dotyczące chmury, umożliwiając skupienie się na określeniu zachowania aplikacji.

Podobnie jak kontenery są przenośnym sposobem definiowania oprogramowania, zasoby Kubernetes zapewniają przenośną definicję działania tego oprogramowania.

Czy Kubernetes zniknie?

Co ciekawe, pomimo emocji, które obecnie budzi Kubernetes, naprawdę możemy o nim zapomnieć w najbliższych latach. Wiele rzeczy, które kiedyś były nowe i rewolucyjne, stanowi obecnie dużą część innych technologii, np: mikroprocesory, mysz, internet.

Kubernetes prawdopodobnie również zniknie i stanie się częścią infrastruktury. Gdy jednak dowiesz się, co musisz wiedzieć, aby wdrożyć aplikację w Kubernetes, będziesz posiadać już podstawową wiedzę.

Przyszłość Kubernetes prawdopodobnie leży w dużej mierze w dziedzinie usług zarządzanych. Wirtualizacja, która była kiedyś ekscytującą nową technologią, teraz stała się po prostu narzędziem. Zamiast uruchamiać własną platformę wirtualizacji, taką jak vSphere lub Hyper-V, większość osób wynajmuje maszyny wirtualne od dostawcy chmury.

Uważamy, że w podobny sposób Kubernetes stanie się standardową częścią innej technologii.

Kubernetes nie załatwia wszystkiego

Czy infrastruktura przyszłości będzie w całości oparta na Kubernetes? Prawdopodobnie nie. Po pierwsze, niektóre rzeczy po prostu nie pasują do Kubernetes (np. bazy danych).

> *Oprogramowanie do orkiestracji kontenerów obejmuje rozwijanie nowych wymiennych instancji bez konieczności koordynacji między nimi. Jednak repliki baz danych nie są wymienne; każda z nich ma unikalny stan, a wdrożenie repliki bazy danych wymaga koordynacji innych węzłów w celu zapewnienia, że zmiany obejmą wszystkie miejsca.*
>
> — Sean Loiselle (*https://www.cockroachlabs.com/blog/kubernetes-state-of-stateful-apps*) (Karaluch Labs)

Chociaż Kubernetes pozwala na uruchamianie prac stanowych (ang. *stateful workloads*), takich jak bazy danych, z niezawodnością klasy korporacyjnej, wymaga to dużego nakładu czasu — więc pozbawione jest sensu (patrz „Uruchom mniej oprogramowania" w rozdziale 3.). Korzystanie z usług zarządzanych jest zwykle bardziej opłacalne.

Po drugie, niektóre zadania nie potrzebują uruchomienia Kubernetes i mogą działać na tzw. platformach bezserwerowych, czyli na zasadzie *funkcja jako usługa* (*FaaS* —ang. *Functions as a Service*).

Funkcje chmurowe i funtainery

Przykładowo AWS Lambda jest platformą FaaS, która pozwala na uruchamianie kodu napisanego w językach Go, Python, Java, Node.js, C # i innych, bez konieczności kompilacji. Amazon robi to wszystko za Ciebie.

Ponieważ rachunek za korzystanie z usługi naliczany jest za każde 100 milisekund, model FaaS jest idealny do obliczeń, które działają tylko wtedy, gdy są potrzebne. Natomiast za wynajęcie serwera w chmurze płacisz raz i działa on przez cały czas, niezależnie od tego, czy go używasz, czy nie.

Takie *funkcje chmurowe* są pod pewnymi względami wygodniejsze niż kontenery (chociaż niektóre platformy FaaS mogą również uruchamiać kontenery). Jednak najlepiej nadają się do krótkich, niezależnych zadań (np. AWS Lambda ogranicza funkcje do 15 minut działania i około 50 MiB dla plików), zwłaszcza tych, które integrują się z istniejącymi usługami obliczeniowymi w chmurze, takimi jak Microsoft Cognitive Services lub interfejs API Google Cloud Vision.

Dlaczego nie lubimy nazywać tego modelu „bezserwerowym"? Ponieważ taki nie jest, gdyż to jest czyjś serwer. Chodzi o to, że nie musisz obsługiwać i utrzymywać tego serwera; dostawca usług chmurowych dba o to za Ciebie.

Nie dla każdego zadania odpowiednie jest działanie na platformach FaaS, ale nadal prawdopodobnie będzie kluczową technologią dla chmurowych aplikacji natywnych w przyszłości.

Funkcje chmurowe nie są ograniczone do publicznych platform FaaS, takich jak Lambda lub Azure Functions: jeśli masz już klaster Kubernetes i chcesz na nim uruchamiać aplikacje FaaS, OpenFaaS (*https://www.openfaas.com/*) i inne projekty open source, jest to możliwe. Taka hybryda funkcji i kontenerów jest czasem nazywana *funtainerami*.

Obecnie jest w fazie rozwoju *Knative*, bardziej zaawansowana platforma dostawy oprogramowania dla Kubernetes, która obejmuje zarówno kontenery, jak i funkcje chmurowe (patrz „Knative" w rozdziale 13.). To bardzo obiecujący projekt, co może oznaczać, że w przyszłości rozróżnienie między kontenerami a funkcjami może całkowicie się zacierać lub zanikać.

Model Cloud Native

Termin *cloud native* staje się coraz bardziej popularny w kontekście mówienia o nowoczesnych aplikacjach i usługach korzystających z chmury, kontenerów oraz orkiestracji, często opartych na oprogramowaniu open source.

Organizacja Cloud Native Computing Foundation (CNCF) (*https://www.cncf.io/*) została założona w 2015 r., aby „wspierać społeczność wokół konstelacji wysokiej jakości projektów, które orkiestrują kontenery jako część architektury mikrousług "(*https://www.cncf.io/about/join/*).

CNCF, część organizacji Linux Foundation, istnieje po to, aby skupiać programistów, użytkowników końcowych i dostawców, w tym głównych dostawców publicznych chmur obliczeniowych. Najbardziej znanym projektem pod patronatem CNCF jest sam Kubernetes, ale fundacja również inkubuje i promuje inne kluczowe elementy rodzimego ekosystemu chmurowego, takie jak Prometheus, Envoy, Helm, Fluentd, gRPC i wiele innych.

Co dokładnie rozumiemy pod pojęciem *cloud native*? Jak większość takich terminów, oznacza to różne rzeczy dla różnych ludzi, ale być może istnieje jakaś wspólna płaszczyzna.

Aplikacje *cloud native* działają w chmurze; to nie jest kontrowersyjne. Jednak samo uruchomienie istniejącej aplikacji w instancji chmury obliczeniowej nie powoduje, że jest ona natywna. Nie chodzi również o uruchamianie w kontenerze lub korzystanie z usług w chmurze, takich jak Azure Cosmos DB lub Google Pub / Sub, chociaż mogą to być ważne aspekty aplikacji typu *cloud native*.

Spójrzmy więc na kilka cech charakterystycznych dla systemów *cloud native*, na które większość ludzi może się zgodzić.

Zdolność do automatyzacji
: Jeśli aplikacje mają być wdrażane i zarządzane przez maszyny, a nie przez ludzi, muszą przestrzegać wspólnych standardów, formatów i interfejsów. Kubernetes zapewnia te standardowe interfejsy w taki sposób, że programiści aplikacji nie muszą się o nie martwić.

Wszechobecność i elastyczność
: Ponieważ są one oddzielone od zasobów fizycznych, takich jak dyski, lub jakiejkolwiek konkretnej wiedzy na temat węzła obliczeniowego, na którym akurat się uruchamiają, mikrousługi w kontenerach można łatwo przenosić z jednego węzła do drugiego, a nawet z jednego klastra do drugiego.

Odporność i skalowalność
: W tradycyjnych aplikacjach występują pojedyncze punkty awarii: aplikacja przestaje działać, jeśli jej główny proces ulegnie awarii, jeśli komputer ulegnie awarii lub przy braku dostępu do zasobów. Aplikacje cloud native są z natury rozproszone, zapewniają wysoką dostępność dzięki redundancji i płynnemu rozkładowi.

Dynamiczność
: Orkiestrator kontenerów, taki jak Kubernetes, może rozplanować użycie kontenerów tak, aby maksymalnie wykorzystać dostępne zasoby. Aby osiągnąć wysoką dostępność, może uruchomić wiele kopii. Może także przeprowadzać ciągłe aktualizacje, aby płynnie aktualizować usługi bez zmniejszania ruchu.

Obserwowalność
: Aplikacje cloud native w swojej naturze są trudniejsze do sprawdzania i debugowania. Zatem kluczowym wymaganiem systemów rozproszonych jest obserwowalność — monitorowanie, rejestrowanie, śledzenie — czyli wszystkie pomiary, które pomagają inżynierom zrozumieć, co robią ich systemy (oraz co robią źle).

Rozproszenie
: Cloud native to podejście do budowania i uruchamiania aplikacji, które wykorzystuje rozproszony i zdecentralizowany charakter chmury. Chodzi o to, jak działa Twoja aplikacja, a nie gdzie działa. Zamiast wdrażać kod jako pojedynczą jednostkę (zwaną *monolitem* — ang. *monolith*), aplikacje cloud native zwykle składają się z wielu współpracujących, rozproszonych *mikrousług*. Mikrousługa to po prostu niezależna usługa, która robi jedną rzecz. Jeśli połączysz wystarczającą liczbę mikrousług, otrzymasz aplikację.

Nie chodzi tylko o mikrousługi

Jednak mikrousługi nie są panaceum. Monolity są łatwiejsze do zrozumienia, ponieważ wszystko jest w jednym miejscu i można śledzić interakcje różnych części. Natomiast trudno skalować monolit zarówno pod względem samego kodu, jak i zespołów programistów, którzy go utrzymują. W miarę wzrostu kodu interakcje między jego różnymi częściami rosną wykładniczo, a system jako całość przestaje być czytelny.

Aplikacja cloud native składa się z mikrousług, ale decydowanie, jakie powinny być te mikrousługi, gdzie są granice i jak różne usługi mają współpracować, nie jest łatwym problemem. Dobry projekt usług cloud native polega na dokonywaniu mądrych wyborów dotyczących dzielenia różnych części architektury. Jednak nawet dobrze zaprojektowana aplikacja cloud native jest nadal systemem rozproszonym, co czyni go z natury złożonym, trudnym do zaobserwowania i uzasadnienia oraz podatnym na awarie.

Chociaż systemy cloud native są zwykle rozproszone, nadal można uruchamiać monolityczne aplikacje w chmurze, używając kontenerów. Może to być krok na drodze do stopniowej migracji części monolitu na zewnątrz do nowoczesnych mikrousług lub chwilowe oczekiwanie na przeprojektowanie systemu, tak aby był w pełni cloud native.

Przyszłość operacji

Operacje, zarządzanie infrastrukturą i administracja systemem to prace wymagające wysokich kwalifikacji. Czy taki personel jest zagrożony z uwagi na przyszłość cloud native? Myślimy, że nie.

Umiejętności te staną się jeszcze ważniejsze. Projektowanie systemów rozproszonych jest trudne. Sieci i orkiestratory kontenerów są skomplikowane. Każdy zespół tworzący aplikacje cloud native

będzie potrzebował umiejętności operacyjnych oraz wiedzy. Automatyzacja uwalnia pracowników od nudnej, powtarzalnej pracy ręcznej, aby radzili sobie z bardziej złożonymi oraz interesującymi problemami, których komputery nie są jeszcze w stanie rozwiązać samodzielnie.

Nie oznacza to, że wszystkie prace związane z zadaniami operacyjnymi są gwarantowane. Sysadmini potrafili sobie radzić bez umiejętności kodowania, z wyjątkiem tworzenia skryptów w powłoce. W środowisku chmurowym to nie zadziała.

W świecie zdefiniowanym przez oprogramowanie umiejętność pisania, rozumienia i utrzymywania oprogramowania staje się krytyczna. Jeśli nie możesz opanować nowych umiejętności lub nie nauczysz się ich, świat Cię porzuci — zawsze tak było.

Rozproszone DevOps

Zamiast koncentrować się w jednym zespole operacyjnym, który obsługuje inne zespoły, wiedza operacyjna zostanie podzielona między wiele zespołów.

Każdy zespół programistów będzie potrzebował co najmniej jednego specjalisty ds. operacyjnych, odpowiedzialnego za stan systemów lub usług świadczonych przez zespół. Będzie także programistą, ale też ekspertem w dziedzinie sieci, Kubernetes, wydajności, odporności oraz narzędzi i systemów, które umożliwią innym programistom dostarczanie kodu do chmury.

Dzięki rewolucji DevOps w większości organizacji nie będzie już miejsca dla programistów, którzy nie mogą realizować zadań operacyjnych, lub operatorów, którzy nie potrafią programować. Różnica między tymi dwoma zawodami szybko znika. Tworzenie i obsługa oprogramowania to zaledwie dwa aspekty tego samego.

Niektóre rzeczy pozostaną scentralizowane

Czy są ograniczenia wynikające ze stosowania DevOps? Może tradycyjny zespół operacji IT zniknie całkowicie, zamieniając się w grupę konsultantów wewnętrznych czy też ekspertów ds. coachingu?

Myślimy, że nie, a przynajmniej nie do końca. Niektóre rzeczy nadal korzystają z centralizacji. Nie ma sensu, aby każdy zespół ds. aplikacji lub usług miał własny sposób wykrywania zdarzeń oraz komunikowania np. o zdarzeniach produkcyjnych, taki jak własny system obsługi klienta czy narzędzia potrzebne do wdrożeń. Nie ma sensu wymyślać czegoś od początku.

Produktywni programiści

Chodzi o to, że samoobsługa ma swoje granice, a celem metodyki DevOps jest przyspieszenie pracy zespołów programistycznych, a nie spowalnianie ich niepotrzebną i zbędną pracą.

Tak, duża część tradycyjnych operacji może i powinna zostać przekazana innym zespołom, przede wszystkim tym, które dotyczą wdrażania kodu i reagowania na zdarzenia związane z kodem. Aby jednak tak się stało, musi istnieć silny, centralny zespół, który wspiera ekosystem DevOps, w jakim działają wszystkie pozostałe zespoły.

Aby nie nazywać tego zespołu operacyjnym, użyjemy nazwy *produktywni programiści* (*DPE* — ang. *developer productivity engineering*). Zespoły DPE robią wszystko, co konieczne, aby pomóc programistom w szybszym wykonywaniu pracy; zajmują się obsługą infrastruktury, narzędziami budowania, problemami z awarią.

I chociaż DPE posiada specjalistyczny zestaw umiejętności, sami inżynierowie mogą przenieść się do organizacji, aby dostarczyć tę wiedzę tam, gdzie jest potrzebna.

Matt Klein inżynier firmy Lyft zasugerował, że chociaż czysty model DevOps ma sens dla startupów i małych firm, to wraz z rozwojem organizacji zachodzi naturalna tendencja przenoszenia ekspertów ds. infrastruktury i niezawodności do zespołu centralnego. Twierdzi jednak, że zespołu nie można skalować w nieskończoność.

> *Do czasu, gdy zespół inżynieryjny osiągnie ~ 75 osób, prawie na pewno działa centralny zespół ds. infrastruktury, który zaczyna budować wspólne właściwości wymagane przez zespoły tworzące mikrousługi. Jednak przychodzi moment, w którym centralny zespół ds. infrastruktury nie może dłużej zarówno budować infrastruktury krytycznej, jak i ją obsługiwać, jednocześnie utrzymując wsparcie dla zespołów ds. produktów w wykonywaniu zadań operacyjnych.*
>
> — Matt Klein (*https://medium.com/@mattklein123/the-human-scalability-of-devops-e36c37d3db6a*)

W tym momencie nie każdy programista może być ekspertem infrastruktury, tak jak pojedynczy zespół ekspertów ds. infrastruktury nie jest w stanie obsłużyć stale rosnącej liczby programistów. W większych organizacjach, mimo że centralny zespół ds. infrastruktury jest nadal potrzebny, istnieje również możliwość włączenia *inżynierów ds. niezawodności* (*SRE* — ang. *site reliability engineer*) do każdego zespołu programistów lub produktów. Wnoszą swoją wiedzę do każdego zespołu jako konsultanci, a także stanowią pomost między rozwojem produktu a operacjami związanymi z infrastrukturą.

Jesteś przyszłością

Jeśli czytasz tę książkę, oznacza to, że będziesz częścią rodziny cloud native. W pozostałych rozdziałach omówimy całą wiedzę i umiejętności, których będziesz potrzebować jako programista lub inżynier operacyjny pracujący z infrastrukturą chmurową, kontenerami oraz Kubernetes.

Niektóre z tych rzeczy będą znane, a niektóre nowe. Mamy jednak nadzieję, że kiedy zapoznasz się z materiałem zawartym w książce, poczujesz się bardziej pewny własnych umiejętności związanych z cloud native. Tak, jest wiele do nauczenia się, ale dasz radę.

Czytaj dalej.

Podsumowanie

Zapoznałeś się z krótką prezentacją chmurowego środowiska DevOps. Mamy nadzieję, że wystarczy ona, abyś nabrał pewności, że będziesz sprawniej rozwiązywał problemy związane z środowiskiem chmurowym, kontenerami oraz Kubernetes.

Poniżej zamieściliśmy krótkie podsumowanie głównych tematów — potem przejdziesz do następnego rozdziału i zapoznasz się z Kubernetes.

- Przetwarzanie w chmurze uwalnia Cię od kosztów zarządzania własnym sprzętem, umożliwiając budowanie odpornych, elastycznych, skalowalnych systemów rozproszonych.
- DevOps zapewnia, że współczesne tworzenie oprogramowania nie kończy się w momencie napisania kodu: chodzi o utworzenie pętli sprzężenia zwrotnego między tymi, którzy piszą kod, a tymi, którzy go używają.
- DevOps wprowadza podejście zorientowane na kod oraz dobre praktyki inżynierii oprogramowania w świecie infrastruktury i operacji.
- Kontenery umożliwiają wdrażanie i uruchamianie oprogramowania w małych, znormalizowanych, samodzielnych jednostkach. To sprawia, że budowanie dużych, różnorodnych, rozproszonych systemów jest łatwiejsze i tańsze dzięki połączeniu mikrousług kontenerowych.
- Systemy orkiestracji zajmują się rozmieszczaniem kontenerów, planowaniem, skalowaniem, tworzeniem sieci i wszystkim, co zrobiłby dobry administrator systemu, ale w sposób zautomatyzowany i programowalny.
- Kubernetes to de facto standardowy system orkiestracji kontenerów — gotowy do użycia już dziś w produkcji.
- Cloud native jest skrótem przydatnym, kiedy mówimy o chmurowych, kontenerowych, rozproszonych systemach, złożonych ze współpracujących mikrousług, dynamicznie zarządzanych przez zautomatyzowaną infrastrukturę w postaci kodu.
- Umiejętności związane z operacjami oraz obsługą infrastruktury, które nie są uważane za przestarzałe w terminologii cloud native, są i będą ważniejsze niż kiedykolwiek.
- Nadal istnieje sens, aby centralny zespół budował i utrzymywał platformy oraz narzędzia, które umożliwiają skorzystanie z metodyki DevOps wszystkim pozostałym zespołom.
- Zaniknie wyraźne rozróżnienie między inżynierami oprogramowania a inżynierami operacji. Od teraz wszystko związane jest z oprogramowaniem i wszyscy jesteśmy inżynierami.

Kubernetes: nowoczesność, skalowalność i prawdziwa niezawodność!

Praca oparta na metodyce DevOps jest w świecie IT rozwiązaniem klasycznym: zespoły właściwie rozumiejące i stosujące te zasady uzyskują dobre efekty tworzenia i wdrażania aplikacji. Rozwój technologii, zwłaszcza chmurowych, wymusza radykalne zmiany nie tylko w funkcjonowaniu systemów informatycznych, ale też w modelu i stylu pracy zespołów programistycznych. Kubernetes jest standardową platformą rozproszonych aplikacji chmurowych. Pozwala na budowanie i wdrażanie niezawodnych, wydajnych i skalowalnych aplikacji. Jednak przebudowa klasycznej infrastruktury IT na rozwiązania chmurowe, aby mogła być przeprowadzona bez dużych utrudnień pracy firmy, wymaga przygotowania i odpowiedzi na kilka ważnych pytań.

W tym wyczerpującym i praktycznym przewodniku opisano, jak działają kontenery Kubernetes, jak je budować i nimi zarządzać oraz jak projektować usługi i infrastrukturę cloud native. Wyjaśniono różnice między budowaniem swoich klastrów a korzystaniem z usług zarządzanych takich firm jak Amazon, Google i Microsoft. Dokładnie przedstawiono — od strony praktycznej — rozwijanie aplikacji, konfigurowanie i obsługę klastrów oraz automatyzację infrastruktury. Wiedza ta pozwoli na sprawne zbudowanie skalowalnej, odpornej na awarie i opłacalnej infrastruktury Kubernetes — w zgodzie z metodyką DevOps.

W książce:

- solidne podstawy działania kontenerów i systemu Kubernetes
- uruchamianie własnych klastrów
- zarządzane usługi Kubernetes od Amazon, Google i innych dostawców
- cykl życia kontenera i optymalizacja klastrów
- najlepsze narzędzia i najnowsze praktyki branżowe
- zasady DevOps w praktyce

John Arundel od trzydziestu lat zajmuje się technologiami informatycznymi, jest konsultantem i autorem kilku książek. Specjalizuje się w tworzeniu infrastruktury cloud native, szczególnie w Kubernetesie. Mieszka w Kornwalii w Anglii.

Justin Domingus jest inżynierem operacyjnym. Zdobył duże doświadczenie w pracy z technologiami chmurowymi w środowiskach DevOps z Kubernetesem. Mieszka w Seattle w stanie Waszyngton.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-283-6927-6

9 788328 369276

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 69,00 zł