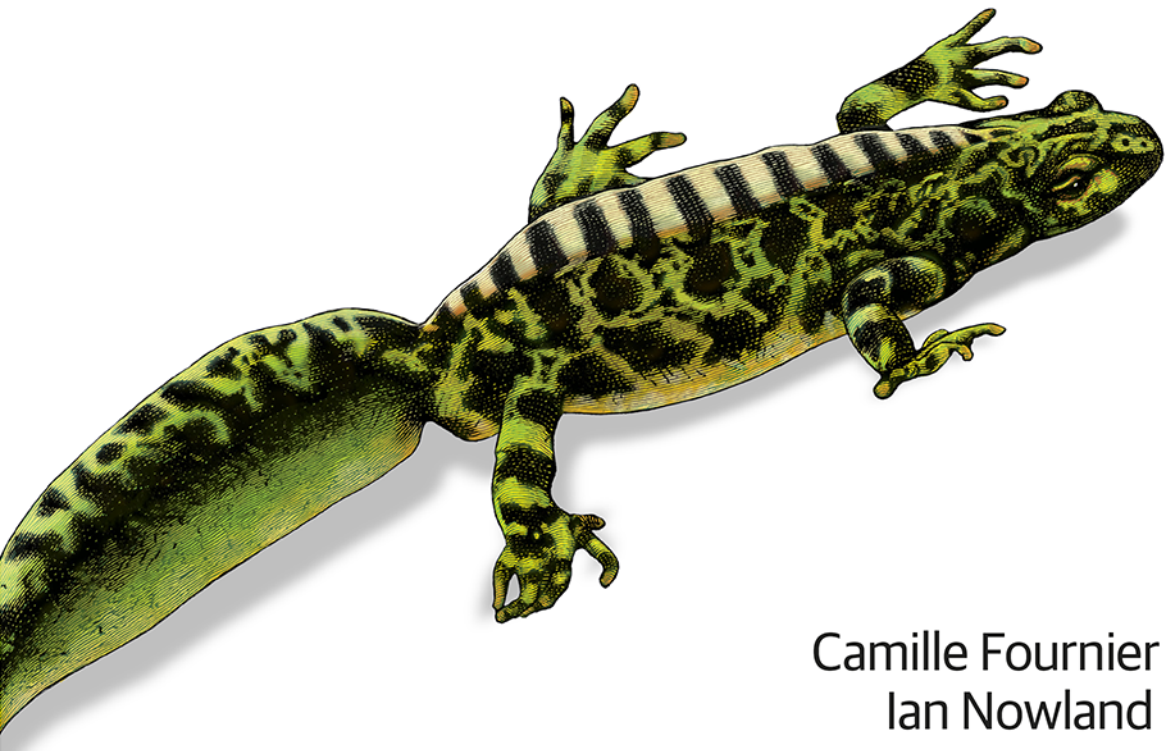
O'REILLY®

Helion

Inżynieria platform dla nowoczesnych liderów

Skuteczne zarządzanie systemami i zespołami



Camille Fournier
Ian Nowland

Tytuł oryginału: Platform Engineering: A Guide for Technical, Product, and People Leaders

Tłumaczenie: Aleksander Łapuć

ISBN: 978-83-289-2555-7

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

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Opinie o książce

Książka ta zawiera wiele praktycznych i szczegółowych porad, podanych w formie doskonale ułożonego i poczytnego przewodnika. Lektura obowiązkowa dla każdego, kto zajmuje się tworzeniem wielkoskalowych systemów.

— Adrian Cockcroft, konsultant ds. technologii w Orionx.net, wcześniej architekt rozwiązań chmurowych w Netflix oraz wiceprezes ds. strategii architektury w AWS

Autorzy przechodzą wprost do sedna, nie zatrzymując się na poziomie frazesów, lecz skupiając się na istotnych zagadnieniach inżynierii platform wewnętrznych. Opisują liczne sposoby, jak omijać typowe pułapki i tworzyć darzone zaufaniem platformy, a nawet uwielbiane — wszystko na bazie własnego bogatego doświadczenia.

— Tanya Reilly, autorka książki Ścieżka kariery inżyniera technicznego

Inżynieria platform to sport zespołowy. A to jest Twój plan gry.

— Kelsey Hightower, były dyrektor w Google (poziom Distinguished Engineer) oraz współautor książki Kubernetes. Tworzenie niezawodnych systemów rozproszonych

Autorzy solidnie argumentują, dlaczego platformy powinny być traktowane jako produkt, bez względu na rozmiar organizacji je wykorzystujących. Podkreślają rolę platformy jako czynnika umożliwiającego odniesienie sukcesu, a nie ogranicznika dostępnych działań.

— Sam Newman, ekspert technologiczny i autor

Poprawne zastosowanie inżynierii platform jest nie lada wyzwaniem: okresy zwrotów z inwestycji wynoszą miesiące lub lata i zwykle powiązanie pomiędzy włożonym wysiłkiem a osiągniętymi wynikami biznesowymi nie jest wprost oczywiste. Camille i Ian zbudowali niejeden wspaniały zespół inżynierii platform, choć często nie było to łatwe — ich doświadczenie przeziera z każdej strony tej fantastycznej książki. Pokazują sposoby budowania zespołów, wyboru właściwych projektów, nawigacji przez meandry polityki organizacyjnej oraz rozpoznania, że osiągnięty został sukces.

— Sarah Wells, niezależna konsultantka i autorka

Camille Fournier i Ian Nowland spokojnie mogliby napisać książkę na temat inżynierii platform — i dokładnie to zrobili. Ich dzieło zawiera wszystkie potrzebne informacje, aby zrozumieć i stosować inżynierię platform. Zdecydowanie polecam.

— James Turnbull, dyrektor ds. technologii i autor

Niezbędna lektura dla każdego lidera, który zamierza zaokrętować się na podróż przez inżynierię platform. Książka stanowi szczegółową mapę, pozwalającą ominąć techniczne, organizacyjne i systemowe wyzwania oraz zaplanować tworzenie wysoce efektywnych platform. Została napisana w oparciu o głęboką znajomość przedmiotu, połączoną z pokorą oraz empatią poznawczą autorów.

— Smruti Patel, wiceprezes ds. inżynierii w Apollo GraphQL

Spis treści

ROZDZIAŁ 1

Dlaczego inżynieria platform staje się niezbędna?

Wszyscy na siebie poupadali: rzepka na dziadka, dziadek na babcię, babcia na wnuczka, wnuczek na Mruczka, Mruczek na Kicię, Kicia na kurkę, kurka na gąskę, gąska na boćka, bociek na żabkę, żabka na kawkę i na ostatku kawka na trawkę.

— Jan Brzechwa, *Rzepka*

Przez ostatnie 25 lat wytwórcy oprogramowania coraz częściej stają wobec problemu, co zrobić z kodem źródłowym, narzędziami i elementami infrastruktury, które są współdzielone przez wiele zespołów. W większości organizacji próbowano rozwiązać ten problem poprzez powoływanie zespołów centralnych, zajmujących się realizacją wszelkich żądań dotyczących współdzielonych komponentów. Niestety zazwyczaj takie podejście nie działało zbyt dobrze. Zespoły centralne były zwykle krytykowane za dostarczanie rozwiązań trudnych w użyciu, ignorowanie potrzeb klientów i stawianie własnych priorytetów ponad priorytetami użytkowników, niewystarczającą stabilność tworzonych systemów, a czasem za wszystkie te przewiny naraz.

W niektórych organizacjach postanowiono w ogóle zarzucić próby usprawnienia działania zespołów centralnych i zlikwidowano je — w zamian oferując każdemu zespołowi aplikacyjnemu bezpośredni dostęp do środowiska chmurowego oraz dowolność w dobieraniu wykorzystywanych komponentów open source. Jednakże takie podejście sprawia, że zespoły mają bezpośrednią styczność ze złożonością operacyjną i utrzymaniową wybranych technologii. W rezultacie nie tylko nie zwiększono wydajności pracy i nie uzyskano znaczących oszczędności, ale nawet w niewielkich zespołach wymuszono zatrudnianie specjalistów DevOps lub niezawodności systemów (SRE). Co gorsza, koszty związane z zarządzaniem złożonością rozwiązań, nawet gdy w zespołach istnieją specjaliści dla tego obszaru, wciąż godzą w produktywność zespołów aplikacyjnych.

Inne z kolei starały się wykorzystać jak najpełniej możliwości rozwiązań chmurowych oraz oprogramowania open source, ale bez likwidacji zespołów centralnych — nie modyfikowano modelu pracy z pełnym przekonaniem, że korzyści przeważą nad wadami. Najlepszym organizacjom udało się pomyślnie stworzyć platformy: wytworzyły współdzielone rozwiązanie

programistyczne, na bazie którego pozostali inżynierowie mogli budować kolejne rozwiązania. Osiągnęły poziom ekspercki w zarządzaniu złożonością środowiska chmurowego oraz oprogramowania open source, dzięki czemu mogły zaoferować swoim użytkownikom wysoki poziom stabilności oraz doprowadzić do współpracy z zespołami aplikacyjnymi na zasadach partnerskich, wsłuchując się w potrzeby i odpowiednio zmieniając rozwiązania centralne. W niektórych organizacjach nazywano ten sposób pracy inżynierią platform, a w niektórych nie. Ale istotą ich działania było zawsze przyjęcie mentalności zorientowanej na rozwiązywanie problemów, rozwijanie odpowiednich ku temu umiejętności oraz stosownie właściwego podejścia — w ten sposób została opanowana wciąż rosnącą złożoność rozwiązań i osiągnięto sukces (wyciągnięcie rzepki) bez wywracania przy okazji wszystkich na trawkę.

Aby przygotować grunt pod dalszą lekturę, w tym rozdziale zajmiemy się kilkoma kluczowymi zagadnieniami:

- Wyjaśnimy, co rozumiemy pod pojęciem platform, oraz kilka innych ważnych terminów, których będziemy używać w dalszej części książki.
- Opiszemy, w jaki sposób architektura systemów stała się znacznie bardziej zagmatwana w erze rozwiązań chmurowych i technologii open source, przez co wszyscy tkwimy w „bagnie nadmiernych uogólnień", powstałym z bezpośredniej ekspozycji na pełną złożoność rozwiązań.
- Wskażemy, w jaki sposób inżynieria platform pozwala zarządzać tą złożonością, a tym samym bezpiecznie opuścić bagno.

W tym rozdziale poruszamy kwestie narzędzi deweloperskich oraz zarządzania infrastrukturą, ale nie martw się, ta książka nie jest wyłącznie dla osób zajmujących się infrastrukturą lub platformami programistycznymi! Posłużymy się systemami typowo używanymi przez programistów, aby przedstawić na przykładach, z czym mamy do czynienia, jednak podstawowe wyzwanie zarządzania złożonością dotyczy rozwoju wszelkiego rodzaju platform wewnętrznych.

Definicja platformy i innych istotnych terminów

Zanim przejdziemy dalej, zdefiniujemy kilka ważnych określeń, z których będziemy korzystać w treści książki. Dzięki temu będziemy się posługiwać tym samym układem odniesienia.

Platforma

Korzystamy z nieco zmodyfikowanej definicji ukutej przez Evana Bottchera w 2018 roku (*https://martinfowler.com/articles/talk-about-platforms.html*). Platforma obejmuje zorganizowane w jedną całość samoobsługowe interfejsy (API), narzędzia, usługi, wiedzę oraz wsparcie użytkowników. W rezultacie powstaje atrakcyjny produkt wewnętrzny, na bazie którego zespoły aplikacyjne[1] mogą łatwo dostarczać funkcjonalności produktu, zwiększając efektywność pracy i zmniejszając potrzebę koordynacji międzyzespołowej.

[1] Czasem będziemy takie zespoły nazywać „użytkownikami" lub „klientami", jeżeli określenia te będą lepiej pasować do kontekstu.

Powstaje zatem pytanie: co właściwie nie jest platformą? Na potrzeby tej książki przyjmujemy, że platformami nazywamy tylko takie produkty, do których powstania i rozwoju niezbędne jest stosowanie inżynierii platform. Zgodnie z tym rozumowaniem dokumentacja na stronie wiki nie jest platformą, ponieważ do jej napisania nie są potrzebne działania inżynierskie. Nie jest także platformą chmura sama w sobie — możesz wykorzystać kilka produktów oferowanych w środowisku chmurowym do wytworzenia platformy wewnętrznej, ale „goła" chmura jest ogromnym zbiorem oferowanych rozwiązań cząstkowych, zbyt rozległym, by stworzyć spójną platformę.

Inżynieria platform

Jest to dyscyplina inżynierii opisująca procesy tworzenia platform i zarządzania nimi. Stosuje się ją, aby okiełznać całkowitą złożoność systemów i w ten sposób umożliwić lepsze wykorzystanie posiadanych sił i środków dla osiągania celów biznesowych. Dzięki stosowaniu w rozwoju platform podejścia opartego na starannym doborze funkcjonalności produktu powstaje programowa warstwa abstrakcji, na bazie której możliwe jest stworzenie szerokiego wachlarza aplikacji biznesowych, wspierających bezpośrednio działania organizacji. Temat ten omówimy szerzej w rozdziale 2.

Efekt dźwigni

Jedną z podstawowych koncepcji inżynierii platform jest wykorzystanie efektu dźwigni — kiedy to praca wykonana przez garstkę inżynierów w obrębie zespołu platformowego redukuje potrzebę ponoszenia znacznie większych nakładów pracy w pozostałych częściach organizacji. Platformy mogą osiągać efekt dźwigni na dwa sposoby: poprzez zwiększenie produktywności inżynierów aplikacji, gdyż mogą oni skupić się na dostarczaniu wartości biznesowej, oraz poprzez poprawienie ogólnej wydajności zespołów inżynierskich, dzięki usunięciu duplikacji zadań w obrębie zespołów aplikacyjnych.

Produkt

Uważamy, że najistotniejsze jest postrzeganie platformy jako produktu. Gdy platformy są postrzegane jako atrakcyjne produkty, oznacza to, że przy wyborze implementowanych funkcji skupiliśmy się na potrzebach klientów — warto zauważyć, że do realizacji tego celu nie wystarczy wyłącznie zatrudnienie menedżerów produktu. Dążymy do osiągnięcia efektu analogicznego do rezultatów osiągniętych przez Steve'a Jobsa dla rodziny produktów Apple: spośród szerokiego zakresu pożądanych funkcjonalności niektóre z nich są starannie i z wyczuciem dobierane do implementacji w produkcie, a inne są świadomie i z rozwagą odrzucane, co często jest nawet ważniejsze.

Bagno nadmiernych uogólnień

Istnieje wiele typów platform wewnętrznych i porady zawarte w tej książce mają zastosowanie do nich wszystkich. Jednakże obecnie najbardziej dotkliwe problemy dostrzegamy w obszarach narzędzi wspierających zarządzanie infrastrukturą oraz prowadzenie prac programistycznych (DevTools). Naszym zdaniem są to czynniki najsilniej zwiększające zainteresowanie inżynierią platform. Wynika to z bardzo bliskiej integracji takich narzędzi z infrastrukturą chmury publicznej oraz z rozwiązaniami open source. Przez ostatnie 25 lat największe zmiany

w branży informatycznej były rezultatem coraz silniejszego wykorzystania chmur publicznych i technologii open source. Jednakże, zamiast usprawnić wytwarzanie rozwiązań i zredukować całkowite koszty właścicielstwa systemów, trendy te spowodowały skutki przeciwne. Dużo łatwiej jest zbudować aplikacje w oparciu o chmurę i komponenty open source, ale znacznie trudniej jest je utrzymywać. W miarę jak systemy stają się coraz większe, odpowiedzialne za nie zespoły pracują coraz wolniej — tak jakby grzęzły w bagnie.

Wszystko sprowadza się do realiów ekonomicznych tworzenia i utrzymywania oprogramowania. Możesz sądzić, że większość kosztów wynika z samego tworzenia oprogramowania, ale w rzeczywistości najkosztowniejsze są wszelkie działania z obszaru utrzymania i operacji[2]. Szacuje się, że co najmniej 60 – 70% kosztów poniesionych w cyklu życia rozwiązania informatycznego przypada po zakończeniu okresu inicjalnego rozwoju, a około jednej czwartej tej sumy przypada wyłącznie na działania migracyjne oraz dostosowywanie istniejących komponentów do zmienionych warunków[3]. Narzut na utrzymanie systemów jest naprawdę duży i obejmuje instalację wymaganych aktualizacji bezpieczeństwa, wielokrotnie ponawiane testowanie rozwiązania, migrację wykorzystywanych komponentów zewnętrznych do nowszych wersji oraz wiele, wiele innych kwestii.

Wykorzystanie chmury i technologii open source tylko pogorszyło problem, ponieważ technologie te dostarczają wciąż rozrastającą się warstwę komponentów i usług **elementarnych**[4] — które są ogólnego przeznaczenia i oferują szeroki zakres funkcjonalności, lecz nie są wzajemnie zintegrowane. Do funkcjonowania komponenty te potrzebują „kleju” — tak określamy kod integrujący te komponenty, narzędzia jednorazowych automatyzacji, konfiguracyjne oraz zarządcze. Klej łączy elementy, ale też utrudnia wprowadzanie późniejszych zmian w rozwiązaniach.

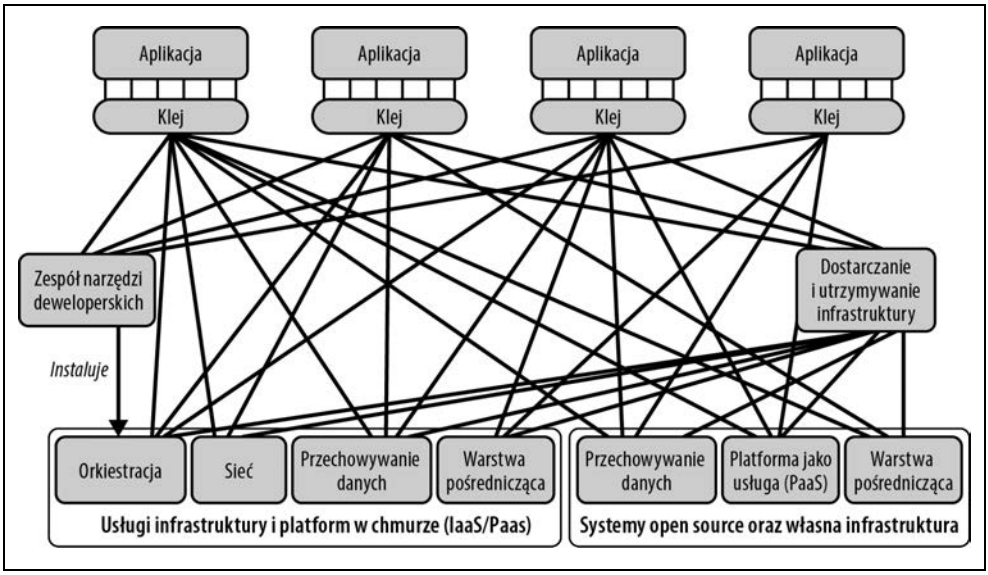
Bagno nadmiernego uogólnienia powstaje w miarę rozprzestrzeniania się kodu integrującego. Każdy zespół aplikacyjny samodzielnie wybiera usługi elementarne, które umożliwią szybkie zbudowanie wymaganego produktu z wykorzystaniem innowacyjnych funkcjonalności. Zespoły samodzielnie wytwarzają wszelkie komponenty kleju niezbędne do szybkiego uruchomienia aplikacji, ponieważ są one nagradzane za jak najszybsze jej dostarczenie. Z czasem, w wyniku powtarzania takiego postępowania, w organizacji powstaje skomplikowany zbiór rozwiązań, którego typową architekturę przedstawia rysunek 1.1.

W przypadku zabagnionej architektury istotą problemu nie są brzydkie i nieuporządkowane schematy, ale to, jak trudno jest przeprowadzić w takim bałaganie jakiekolwiek modyfikacje. Możliwość dokonywania ciągłych zmian w aplikacjach jest bardzo ważna, ewolucja jest wymuszana wprowadzaniem nowych funkcjonalności albo powstaniem nowych wymagań operacyjnych. Każdy komponent open source oraz elementarna usługa chmurowa także podlegają

[2] Bardzo dobry przykład diagramu przedstawiającego cykl życia oprogramowania możesz znaleźć tutaj: *https://emauton.org/2016/03/22/a-software-lifecycle-update/*.

[3] Polecamy artykuł Jussiego Koskinena na temat kosztów utrzymania oprogramowania: *https://ocw.unican.es/pluginfile.php/2171/course/section/2000/SMCOSTS.pdf*.

[4] W opisie wizji chmury AWS są one dosłownie nazywane „usługami prymitywnymi” (ang. *primitives*): *https://www.aboutamazon.com/news/company-news/amazon-ceo-andy-jassy-2023-letter-to-shareholders*.

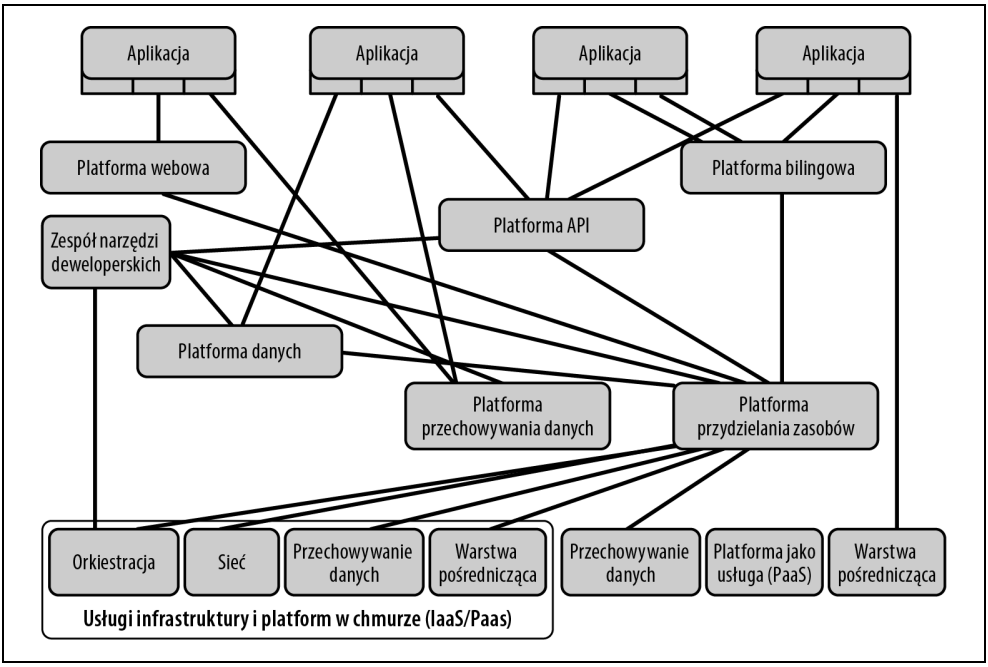


Rysunek 1.1. Bagno nadmiernych uogólnień połączone za pomocą kleju

regularnym zmianom, co wymusza aktualizację kodu integracyjnego po stronie aplikacji. W przypadku gdy kod łączący komponenty występuje wszędzie, nawet stosunkowo prosta aktualizacja usługi elementarnej (na przykład instalacja poprawki bezpieczeństwa) staje się gigantyczną inicjatywą, która dotyka każdego obszaru organizacji, wymuszając modyfikacje kodu integracyjnego wielu aplikacji i przeprowadzenie intensywnego testowania — koszt taki można porównać do haraczu płaconego produktywnością wszystkich zespołów.

Aby uniknąć takich sytuacji, należy znacząco ograniczyć ilość wykorzystywanego kleju, co sprowadza się do starej zasady architektonicznej „więcej elementów, mniej połączeń". Wykorzystanie platform ucieleśnia tę zasadę i pozwala wydostać się z bagna. Tworząc platformę, wprowadzasz warstwę abstrakcji, ograniczoną do dobrze określonego zestawu technologii i komponentów oraz dostosowaną do konkretnych potrzeb organizacyjnych. Z kolei różne potrzeby organizacyjne możesz odseparować od siebie, tworząc dla nich oddzielne warstwy abstrakcji. W rezultacie Twoja architektura zaczyna przypominać tę z rysunku 1.2.

Podsumowując: platformy ograniczają wymaganą ilość kodu integrującego poprzez wprowadzenie abstrakcji wybranych rozwiązań technicznych oraz udostępnianie hermetyzowanych interfejsów, ukrywających przed użytkownikami złożoność pewnych obszarów (także gdy implementacja wewnętrzna tych obszarów ulega zmianie). Są to koncepcje istniejące w informatyce od zarania dziejów i gruntownie poznane — powstaje więc pytanie: po co wprowadzać inżynierię platform? Zanim na nie odpowiemy, przedstawimy zmiany, jakie zaszły w ciągu ostatniego ćwierćwiecza w obszarze inżynierii oprogramowania dla przedsiębiorstw.



Rysunek 1.2. Wpływ wprowadzenia platform na ilość kleju

Jak utknęliśmy w bagnie nadmiernego uogólnienia?

W branży informatycznej przez ostatnie 25 lat, w wyniku upowszechnienia dostępu do internetu, doszło do gigantycznych przeobrażeń. Jeżeli pracujesz w tym zawodzie wystarczająco długo, to wiesz doskonale, jakim zmianom uległy wszystkie aspekty produkcji oprogramowania. Ale jeżeli stosunkowo niedawno dołączyłeś do branży, to wiedz, że, bez cienia przesady, powstanie bagna nadmiernego uogólnienia wynika w przeważającej mierze z samego istnienia internetu oraz z presji ciążącej na dostarczaniu aplikacji: coraz więcej, coraz szybciej i bez błędów. Przyjrzymy się teraz kluczowym przyczynom dokonanego skrętu na manowce oraz implikacjom z tego wynikającym.

Przyczyna nr 1: eksplozja możliwości

Rozwój internetu wytworzył niesamowicie wielki popyt na nowe oprogramowanie, a każdy program musi być uruchamiany na jakimś urządzeniu, niezależnie od tego, co określenie „przetwarzanie bezserwerowe" (ang. *severless*) wydaje się sugerować. Początkowa fala zwiększonej liczby aplikacji spowodowała instalowanie dodatkowych serwerów w centrach danych, co doprowadziło do rozwoju inżynierii infrastruktury. Każde przedsiębiorstwo kupowało coraz więcej komputerów i elementów sieciowych, ciągły rozwój wymagał prowadzenia intensywnych negocjacji z dostawcami centrów danych, gigantyczne ilości wyposażenia były instalowane jak świat długi i szeroki — infrastruktura przez wielkie „I" zajmowała się inżynierią przez wielkie „I", aby napędzać internet przez wielkie „I".

Nie chcemy umniejszać wagi trudności, jakie zostały przezwyciężone w tym stosunkowo krótkim okresie. Jednakże twórcy aplikacji byli nieustannie sfrustrowani, bo konieczne były ciągłe interakcje z zespołami infrastruktury oraz bezpośrednie zajmowanie się problemami w zarządzaniu sprzętem. Bolączkami zespołów aplikacyjnych były ograniczony i wciąż zmieniający się wybór serwerów, częste problemy ze zdolnościami operacyjnymi centrów danych oraz dziwne problemy operacyjne związane z działaniem sprzętu, których nikt nie mógł zdiagnozować (typowy komentarz dla tego typu usterek brzmiał: „nic nie ma w logach systemowych, to musi być problem w waszym kodzie").

Nie dziwi zatem, że gdy tylko pojawiły się publiczne usługi chmurowe, sfrustrowani programiści aplikacyjni z chęcią przenieśli się do świata, w którym pozornie mogli kontrolować swój los za pomocą wywołanych operacji API. Nawet duże przedsiębiorstwa, zachowawczo podchodzące do zmian, w jakimś stopniu zaczęły wykorzystywać rozwiązania chmurowe — i to pomimo istnienia sensownych zastrzeżeń dotyczących złożoności architektonicznej takich rozwiązań, zagrożeń bezpieczeństwa z nich wynikających, ich niezawodności oraz kosztów.

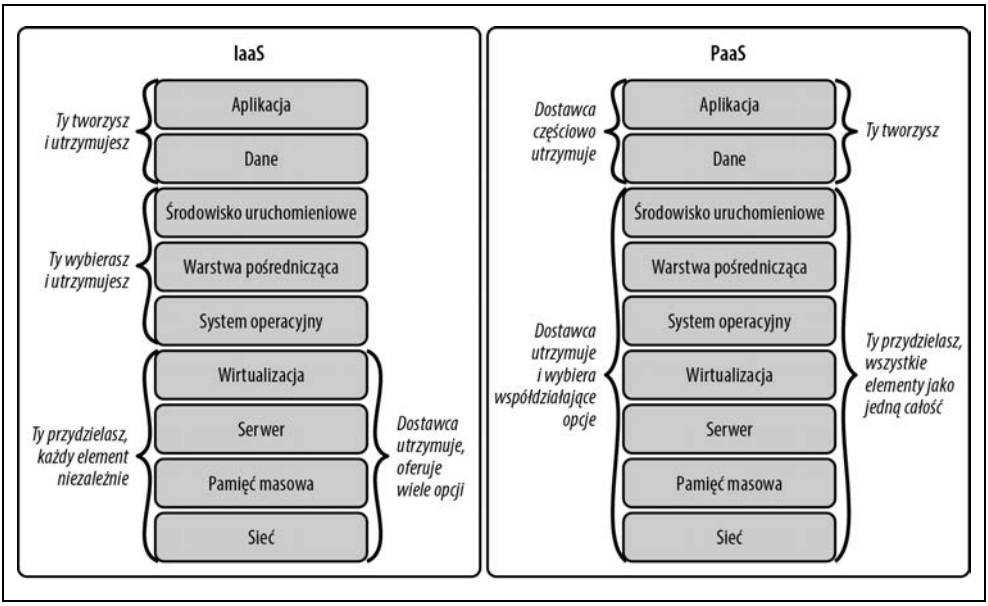
Niestety okazało się, że nie tylko te sensowne zastrzeżenia były prawdziwe, ale wręcz rzeczywista sytuacja wyglądała gorzej, niż się obawiano. Początkowo przejście do chmury opierało się na obietnicy dostarczania platform jako usług (PaaS), dzięki czemu aplikacje miały stać się niezależne od warstwy infrastruktury. Jednakże zwykle stosowane dostarczanie infrastruktury jako usługi (IaaS) spowodowało jeszcze ściślejsze powiązanie aplikacji z infrastrukturą, niż było to spotykane wcześniej. Ważne, żebyś zapamiętał różnicę między tymi rodzajami usług:

- W przypadku **infrastruktury jako usługi** (ang. *Infrastructure as a Service*, IaaS) dostawca udostępnia API, za pomocą którego możliwe jest tworzenie środowiska wirtualnych maszyn, wraz z innymi infrastrukturalnymi usługami elementarnymi, w obrębie którego uruchamiana jest aplikacja, w dużej mierze w taki sam sposób, jak miałoby to miejsce w środowisku fizycznych serwerów.
- W przypadku **platformy jako usługi** (ang. *Platform as a Service*, PaaS) dostawca przejmuje pełną odpowiedzialność za działanie i utrzymywanie infrastruktury niezbędnej dla aplikacji, czyli zamiast oferować usługi elementarne zapewnia komponenty wyższego poziomu abstrakcji, umożliwiając uruchamianie aplikacji w skalowalnym środowisku izolowanym.

Na rysunku 1.3 przedstawiliśmy ogólne porównanie obu rodzajów usług.

Początkowo w modelu PaaS pokładano duże nadzieje, licząc, że zespoły aplikacyjne będą mogły wykorzystać w pełni dojrzałe rozwiązania, tak łatwe w użyciu jak platforma Heroku, ale oferujące wsparcie dla bardziej złożonych aplikacji[5]. Niestety te platformy nie były w stanie wspierać szerokiego zakresu aplikacji ani nie oferowały możliwości integracji z istniejącymi aplikacjami lub infrastrukturą. Dlatego prawie wszystkie przedsiębiorstwa na dużą skalę

[5] Innymi platformami w modelu PaaS, które nie znalazły szerokiego uznania, były Force.com, AWS Elastic Beanstalk oraz Google AppEngine. Obecnie dostawcy usług często określają mianem PaaS bardziej elastyczne oferty, co oznacza, że ich wykorzystanie wymaga powiązania z innymi usługami IaaS, więc problemy ze złożonością pozostają nierozwiązane.



Rysunek 1.3. Porównanie rozkładu odpowiedzialności pomiędzy dostawcę i klienta w modelach usługowych IaaS i PaaS

produkujące oprogramowanie na własny użytek wykorzystują model IaaS i na oferowanych elementach uruchamiają własne aplikacje, godząc się na zwiększoną złożoność i bardziej skomplikowane utrzymanie infrastruktury, a uzyskując w zamian możliwości elastycznego dokonywania wyborów.

Rozwój systemu orkiestracji Kubernetes jest de facto przyznaniem, że oba modele, PaaS i IaaS, nie spełniły pokładanych w nich oczekiwań. Jest to próba uproszczenia środowiska IaaS poprzez wymuszanie projektowania aplikacji jako natywnych dla chmury, co redukuje ilość niezbędnego kleju związanego z infrastrukturą. Jednakże, pomimo wprowadzania standaryzacji złożonych komponentów, sam problem złożoności nie został rozwiązany. Kubernetes jest warstwą pośredniczącą, która stara się wspierać możliwie wiele różnych konfiguracji obliczeniowych, ale jednocześnie stanowi typową „przeciekającą abstrakcję", wymuszającą na użytkowniku zajmowanie się zbyt wieloma szczegółami tych konfiguracji. Owszem, większość kodu integracyjnego dołączonego do aplikacji jest w postaci plików YAML, a mniejsza część ma postać skryptów Terraform[6] — ale, jak już wskazywaliśmy, celem stosowania inżynierii platform jest redukcja całkowitej wymaganej ilości kleju.

Kubernetes stanowi także przykład drugiego źródła złożoności rozwiązań, o którym wspominaliśmy. Wraz z rozwojem usług chmurowych następował rozwój wszelkiego typu ekosystemów oprogramowania open source. Dawniej trzeba było zapłacić dostawcy za dostarczenie narzędzi deweloperskich i rozwiązań integracyjnych. Obecnie istnieje mnóstwo świetnie rozwijających się ekosystemów, które oferują szeroki wybór narzędzi deweloperskich, bibliotek,

[6] Więcej na ten temat piszemy w rozdziale 2.

a nawet w pełni niezależnych systemów, takich jak Kubernetes. Taka ilość możliwości jest utrudnieniem w efektywnej pracy. Zespoły aplikacyjne zwykle potrafią znaleźć rozwiązanie open source, które jest optymalne dla ich zastosowań, ale niekoniecznie dla jakiegokolwiek innego zespołu w organizacji. Wybór dokonany z uwzględnieniem potrzeb tylko tej jednej aplikacji skutkuje błyskawicznym dostarczeniem jej początkowej wersji. Jednakże ostatecznie zmienia się w trudne do niesienia brzemię, ponieważ zespół musi samodzielnie ponosić wszystkie koszty wynikające z wybranego przez niego — pozornie darmowego[7] — komponentu open source.

Przyczyna nr 2: zwiększone wymagania operacyjne

Równolegle z eksplozją liczby infrastrukturalnych usług elementarnych oraz liczby opartych na nich aplikacji nikt nie zadał pytania, kto będzie je utrzymywał oraz w jaki sposób. W latach dziewięćdziesiątych, zanim nastąpił intensywny rozwój internetu, firmy wytwarzające oprogramowanie na własne potrzeby zazwyczaj definiowały dwie role, a osoby pełniące te role były przeważnie zatrudniane w kompletnie oddzielnych zespołach:

Deweloper oprogramowania

: Odpowiedzialny za określenie architektury aplikacji, zakodowanie i przetestowanie rozwiązania. W efekcie jego pracy aplikacje były wytwarzane jako monolityczne pakiety oprogramowania przekazywane do innych zespołów w celu uruchomienia i wykorzystania.

Administrator systemów

: Odpowiedzialny za wszelkie kwestie związane z działaniem oprogramowania (zbudowanego wewnątrz organizacji, dostarczonego przez dostawcę lub zainstalowanego jako oprogramowanie open source) uruchamianego produkcyjnie na zasobach sprzętowych organizacji.

W miarę rozwoju internetu sukces przedsiębiorstw w coraz większym stopniu opierał się na wykorzystaniu oprogramowania wytworzonego na własne potrzeby, a definicja tych ról zaczęła się zmieniać. Coraz więcej aplikacji wymagało zapewnienia wsparcia operacyjnego przez całą dobę przez siedem dni w tygodniu, co doprowadziło do rozrostu zespołów zatrudniających *inżynierów utrzymania*. Zwykle role administratorów systemu pełnili niedoświadczeni pracownicy, dopiero rozpoczynający karierę w informatyce — praca taka była traktowana jako poligon, a po wykazaniu się w tej roli mogli oni liczyć na awans do ról mniej związanych z utrzymaniem.

Niektóre przedsiębiorstwa wciąż przewidują w swojej strukturze istnienie zespołów utrzymania, ale ich znaczenie jest coraz mniejsze. Na początku obecnego stulecia deweloperzy oprogramowania zaczęli przyjmować „zwinny" model pracy, oparty na regularnym tworzeniu wydań aplikacji dostarczających przyrostowo wymagane funkcjonalności, gdyż było to postrzegane jako skuteczniejsza metoda pozyskiwania informacji zwrotnej, a zatem dostarczania lepszego produktu. Metody zwinne stanowiły istotne wyzwanie dla modelu opartego na wydzieleniu

[7] Scott McNealy, były prezes Sun Microsystems, obrazowo porównywał wykorzystywanie oprogramowania open source do otrzymania szczeniaka: pozornie jest darmowe, ale w długiej perspektywie czasowej generuje koszty (*https://www.zdnet.com/article/open-source-is-free-like-a-puppy-is-free-says-sun-boss-3039202713/*).

zespołów utrzymania: dochodziło do pewnych napięć pomiędzy zespołami, z których jeden był rozliczany z jak najszybszego wprowadzania zmian do kodu aplikacji, a drugi miał bezpośrednio poprawiać wszelkie problemy związane z działaniem tego kodu w środowisku produkcyjnym. Przy czym określenie „pewne napięcia" jest mocnym niedopowiedzeniem. Niektórzy z nas pamiętają z własnego doświadczenia, jak intensywnie obie strony potrafiły obwiniać się nawzajem, zwłaszcza gdy nastąpiła jakaś poważniejsza awaria związana z wprowadzaniem zmian w produkcie. Główną trudnością był brak jasnego podziału odpowiedzialności, ponieważ metody zwinne zacierały granice między rozwojem a utrzymaniem.

W efekcie opracowano i zaczęto dosyć szeroko stosować model współpracy mający na celu połączenie rozwoju aplikacji z działaniami utrzymaniowymi, nazywany *DevOps*. Dość szybko zaczęto kojarzyć stosowanie tego modelu nie tylko ze zmianą kultury pracy, ale także ze stosowaniem określonych technologii i powoływaniem konkretnych ról w organizacjach. Ale mimo wszystkich pomysłów na połączenie światów wciąż trzeba było wykonywać zadania utrzymaniowo-operacyjne, więc firmy stosowały jedną z dwóch strategii:

Rozdzielone zespoły

Zachowano rozdział na niezależne zespoły utrzymaniowe i deweloperskie, ale do obowiązków zespołów utrzymaniowych dodano część zakresu prac programistycznych, w szczególności dodane zostało tworzenie skryptów i narzędzi potrzebnych do instalowania aplikacji na środowiskach produkcyjnych oraz integrujących aplikację z infrastrukturą. W ten sposób wcześniejsze zespoły operacyjne, złożone z inżynierów utrzymania, stały się nowymi zespołami DevOps, złożonymi z inżynierów DevOps.

Połączone zespoły

Zespoły utrzymaniowy i deweloperski zostały połączone w jedność. Podsumowaniem tego podejścia jest stwierdzenie „kto buduje, ten utrzymuje", co oznaczało, że każda osoba pracująca przy systemie była członkiem jednego i tego samego zespołu. Każdy członek zespołu miał pewną ilość zadań utrzymaniowych do wykonania (w tym najbardziej istotne rotacyjne dyżury). Część zespołów odnosiła sukcesy, zatrudniając wyłącznie programistów, podczas gdy inne były tworzone jako zespoły wielofunkcyjne, których członkami byli też specjaliści odpowiedzialni za tworzenie skryptów i narzędzi potrzebnych do instalowania aplikacji w środowiskach produkcyjnych oraz integrujących aplikację z infrastrukturą. W niektórych przedsiębiorstwach takie osoby także nazywano inżynierami DevOps[8].

Mniej więcej w tym samym czasie, około 2004 roku, firma Google odeszła zupełnie od koncepcji zespołów administratorów systemów na rzecz podejścia określanego jako **inżynieria niezawodności systemów** (ang. *site reliability engineering*, SRE). W 2015 roku, gdy koncepcja DevOps sięgała szczytów swojej popularności, wydana została książka opisująca praktyki stosowane przez Google'a: *Site Reliability Engineering. Jak Google zarządza systemami produkcyjnymi*. Publikacja wywołała spore poruszenie, ponieważ całkiem wiele przedsiębiorstw wdrażających koncepcję DevOps napotykało trudności z efektywnym jej stosowaniem. Podejście

[8] Z kolei w innych przedsiębiorstwach nazywano ich inżynierami systemowymi lub inżynierami ds. rozwoju systemów.

SRE kładzie bardzo silny nacisk na stosowanie procesów ukierunkowanych na zapewnianie stabilności oraz określanie obszarów odpowiedzialności w organizacji, więc wiele osób uważało, że jest to cudowny lek na wszystkie bolączki branży, dzięki któremu uda się doskonale zrównoważyć potrzeby administracyjne i funkcjonalne, co z kolei doprowadzi do tworzenia znacznie bardziej stabilnych systemów.

Naszym zdaniem koncepcja SRE w swojej oryginalnej formie nie odniosła znaczącego sukcesu poza firmą Google. Definiowane procesy były zbyt rozbudowane, a ich pomyślne wykonanie w zbyt wielkim stopniu opierało się na specyficznej kulturze i organizacji pracy, które zostały wytworzone w Google'u, firmie tworzącej największą na świecie wyszukiwarkę. Świetnie podsumował to Dave O'Connor, były dyrektor ds. SRE w Google'u, który na podstawie własnych doświadczeń w pracy poza Google'em napisał w 2023 roku artykuł wskazujący sześć powodów, żeby nie powoływać zespołu SRE (*https://log.andvari.net/6reasons.html*). Stwierdził on: „Kolejnym etapem rozwoju branży produkującej oprogramowanie powinno być całkowite usunięcie barier pomiędzy zespołami SRE i zespołami wytwarzającymi aplikacje, aby dokonywanie racjonalnych inwestycji w stabilność rozwiązań, z uwzględnieniem konkretnych potrzeb, stało się wszechobecnym nastawieniem".

Nie da się uniknąć uwzględniania potrzeb operacyjnych przy produkcji oprogramowania. Każde przedsiębiorstwo, które oferuje dostęp do swoich aplikacji poprzez internet, musi zapewniać wsparcie operacyjne tych aplikacji w okresach, gdy jest ona wykorzystywana przez użytkowników (czasem całodobowo, czasem w godzinach roboczych, a czasem coś pomiędzy). Powstaje pytanie, jak zarządzać tymi potrzebami w możliwie ekonomiczny i stabilny sposób. Na pewno trzeba ograniczyć liczbę wymaganych zespołów administratorskich (czyli, zgodnie z opisaną wcześniej terminologią, rozdzielonych zespołów DevOps/SRE) oraz jak najbardziej ułatwić zespołom rozwojowym samodzielne instalowanie i utrzymywanie oprogramowania — na tej wizji opierała się oryginalna koncepcja DevOps.

Skutek: grzęźniemy w bagnie

Sprawa wygląda tak: mamy coraz więcej zespołów aplikacyjnych, które podejmują coraz więcej decyzji wobec coraz bardziej złożonego zestawu wykorzystywanych systemów open source i chmurowych usług elementarnych. Dzieje się tak, ponieważ zespoły aplikacyjne chcą dostarczać produkty programistyczne jak najszybciej, więc potrzebują jak najszybciej rozwiązywać problemy, które przed nimi stają — a najlepszą strategią na to jest wykorzystywanie systemów najlepiej dopasowanych do natury postawionych zadań (lub systemów najlepiej znanych zespołom). No i jeżeli zespoły mają wziąć pełną odpowiedzialność operacyjną za rozwiązanie, to nie dziwne, że chcą same wybrać, czym się będą truć!

Trzeba też pamiętać, że inżynierowie dodający nowe funkcjonalności do aplikacji nie są jedyną grupą, której zależy na jak najszybszym instalowaniu zmian. Zwiększająca się dostępność rozwiązań poprzez internet spowodowała znaczny rozrost powierzchni ataku, prowadząc do coraz liczniejszych incydentów bezpieczeństwa oraz wciąż odkrywanych nowych przykładów podatności na zagrożenia. Rezultatem jest coraz szybsze tempo zmian wprowadzanych do elementów infrastruktury i komponentów open source, aby zapobiegać tym zagrożeniom. Niektóre

systemy lub komponenty zwiększyły częstotliwość dostarczania nowych wersji, skracając odstęp między kolejnymi wydaniami z lat do miesięcy. Każda opublikowana wersja oznacza nawał pracy dla zespołów aplikacyjnych, które muszą nie tylko zaktualizować swój kod integrujący, ale także przetestować rozwiązanie, a może nawet zaplanować migrację pozostałych komponentów.

Nacisk na wprowadzanie zmian doprowadził do powstania bagnistego bałaganu kodu integrującego, obarczonego dodatkowo długofalowymi konsekwencjami wyborów podjętych przez poszczególne zespoły. Każdy nowo tworzony komponent oprogramowania powoduje podejmowanie kolejnych decyzji i powstawanie dodatkowego kodu integrującego, tylko pogarszając warunki na tym grzęzawisku złożoności i zamykając kolejnych deweloperów w lepkiej pułapce. Nawigacja poprzez taki podmokły teren jest trudna, przedzieranie się przezeń powolne, a dodatkowo na każdym kroku można napotkać żarłoczne operacyjne aligatory (lub, co gorsza, krokodyle!). Jak można wydostać się z takiego ambarasu? Nie powinno Cię zaskoczyć, że naszym zdaniem ratunkiem jest stosowanie inżynierii platform, a za chwilę opiszemy, jak z tej pomocy skorzystać.

W jaki sposób inżynieria platform pozwala oczyścić bagno?

Jeżeli tkwisz w bagnie nadmiernych uogólnień, to warto docenić intelektualny powab inżynierii platform. Aktualnie zatrudniasz coraz więcej osób w zespołach infrastruktury, DevTools, DevOps i SRE, ale wciąż nie możesz nadążyć za zwiększającą się złożonością, pochodzącą z systemów open source i rozwiązań chmurowych. Twoje aplikacje stają się coraz bardziej skomplikowane, deweloperzy aplikacji pracują coraz mniej wydajnie, a Ty potrzebujesz wyjścia z tej sytuacji. Tworzenie platform, aby okiełznać istniejącą złożoność, wygląda naprawdę zachęcająco.

Ale żeby zacząć budować platformy, należy poczynić istotne nakłady inwestycyjne. Chodzi nie tylko o bezpośrednie koszty, wynikające z budowy i utrzymania platform, ale także dodatkowy narzut pracy, związany z ograniczeniem zespołom aplikacyjnym swobody wyboru komponentów programowych i chmurowych usług elementarnych. Ponadto powołanie zespołu inżynierii platform może wymusić dodatkowe koszty organizacyjne w postaci restrukturyzacji, przekwalifikowywania pracowników między rolami oraz wprowadzenia nowego obszaru zainteresowania w przedsiębiorstwie. W tym rozdziale omówimy, w jaki sposób długofalowe korzyści ze stosowania platform i ich inżynierii mogą uzasadnić konieczność poniesienia tych początkowych nakładów.

Ograniczenie dostępnych usług elementarnych i zmniejszenie dodatkowego narzutu integracyjnego

Istnienie przeogromnego bogactwa wyboru nie jest wyłącznie złem: dzięki niemu nowo tworzone aplikacje mogą być dostarczane znacznie szybciej niż kiedyś, a deweloperzy aplikacyjni mają więcej autonomii i poczucia właścicielstwa, gdyż czerpią radość z rozwoju swoich

systemów. Te korzyści często idą w zapomnienie, gdy przedsiębiorstwa zaczynają poszukiwać sposobów na obniżenie kosztów utrzymania systemów i ograniczenie długofalowych negatywnych konsekwencji istnienia różnorodnych opcji do wyboru. Często pierwszym instynktownym działaniem kierownictwa jest autorytarne ustalenie zestawu wymaganych standardów. Zwykle padają kwestie podobne do tej: „Jestem ekspertem w dziedzinie baz danych, więc wybiorę technologie bazodanowe, z których wy, zespoły aplikacyjne, będziecie mogli korzystać". Albo: „Jestem architektem, więc wybiorę wszystkie narzędzia programistyczne i pakiety oprogramowania". Albo: „Jestem dyrektorem pionu techniki, więc zadecyduję o wszystkim". Nieuchronnie takim ekspertom będzie bardzo trudno zrozumieć wystarczająco dogłębnie wszelkie potrzeby biznesowe, aby podjąć optymalne decyzje, a ucierpią zespoły aplikacyjne. Standaryzacja ogłaszana z pozycji autorytetu nie jest wystarczająco dobrym rozwiązaniem.

W ramach inżynierii platform uznajemy, że współczesne zespoły inżynierskie powinny czerpać radość z pracy z wykorzystywanymi systemami. Zespoły dostarczające takie systemy powinny być wrażliwe na potrzeby użytkowników, traktując ich jak klientów, a nie działać wyłącznie z nastawieniem na redukcję kosztów lub ograniczanie wysiłku utrzymania systemów. Zamiast ogłaszania z pozycji autorytetu listy wymaganych standardów w ramach inżynierii platform stosowane jest podejście zorientowane na klienta, nastawione na staranne dobieranie niewielkiego zestawu usług elementarnych, które pozwalają sprostać szerokiemu wachlarzowi wymagań. Przy takim sposobie pracy wymagane jest wypracowywanie kompromisów uwzględniających realia biznesowe, konieczność przyrostowego budowania platform o przemyślanej architekturze oraz informację zwrotną pozyskiwaną od zespołów aplikacyjnych w bezpośrednim partnerskim dialogu. Prawidłowo wdrożony sposób pracy sprawi, że rozwiązania dostępne dzięki platformom będą w naturalny sposób preferowane, gdyż ich zalety będą jasne dla wszystkich partnerów uczestniczących procesie — nie będzie potrzeby odwoływania się do autorytetu architekta, administratora baz danych, dyrektora pionu technologii lub wiceprezesa ds. platform. W ten sposób możesz ograniczyć liczbę wykorzystywanych komponentów open source oraz chmurowych usług elementarnych bez doświadczania najgorszych konsekwencji towarzyszących standardom ustalanym nakazowo.

Redukcja kodu integrującego w obrębie aplikacji

Inżynieria platform ma na celu nie tylko zmniejszenie liczby wykorzystywanych usług elementarnych, ale także redukcję kodu integrującego potrzebnego do ich obsługi. Usunięcie większości kleju po stronie aplikacji jest możliwe dzięki tworzeniu w obrębie platform systemowych warstw abstrakcji, przykrywających usługi elementarne i będących w stanie sprostać szerszym potrzebom niż pojedyncza usługa. Najlepiej będzie pokazać tę koncepcję na przykładzie, więc przeanalizujemy typowo spotykane wyzwanie z zakresu zarządzania infrastrukturą z wykorzystaniem narzędzia Terraform.

Złożoność w systemach open source i usługach chmurowych objawia się na wiele sposobów, a jednym z najbardziej kosztownych jest ich konfiguracja — niekończące się listy parametrów, które trzeba ustawić, a błędne ustawienie nawet jednego z nich prowadzi nieuchronnie do problemów w środowisku produkcyjnym. Najbardziej dotkliwie takie błędy objawiają się w konfiguracji środowisk chmurowych, a narzędziem stworzonym specjalnie po to, by przeciwdziałać

problemom tego typu, jest system Terraform, który umożliwia konfigurowanie infrastruktury jako kodu źródłowego (ang. *infrastructure as a code*, IaC). Według stanu na rok 2024 jest to najlepsze tego typu narzędzie, stanowiące doskonały przykład tego, jak w ramach inżynierii platform można poradzić sobie z niedogodnościami kodu integrującego.

Początkowo zespoły aplikacyjne zachwyciły się dostępnym szerokim wyborem różnorodnych opcji w menu dostawców usług IaaS, a w większości przedsiębiorstw uznano, że najłatwiej będzie, gdy każdy zespół będzie samodzielnie odpowiadał za wytworzenie i utrzymywanie elementów infrastruktury chmurowej w ramach konfiguracji swojej własnej aplikacji. W praktyce oznaczało to tyle, że każdy zespół aplikacyjny stał się po części zespołem inżynierów rozwiązań chmurowych, zdobywając doświadczenie w zakresie zarządzania konfiguracją i zasobami infrastrukturalnymi. Zarządzanie konfiguracją i infrastrukturą musi być wspierane wykorzystaniem narzędzi i szablonów, takich jak Terraform, gdyż inaczej nie jest możliwe uzyskanie powtarzalnej, odtwarzalnej, bezpiecznej i weryfikowalnej infrastruktury. Wobec tego większość zespołów aplikacyjnych zaczęła dokształcać się w zakresie wykorzystywania tego narzędzia. Zaobserwowaliśmy, że zwykle rozwój takich zespołów można podsumować w postaci kilku etapów:

1. Inżynierowie przeważnie niezbyt chętnie uczą się obsługi zupełnie nowych zestawów narzędzi, gdy są one wykorzystywane do niezbyt często wykonywanych zadań. Konfiguracja i udostępnianie elementów infrastruktury nie zdarzają się każdego dnia — nawet w zespołach, które mają bardzo dojrzałe procesy testowania odporności i regularnie budują swoje systemy od zera. Z czasem więc zadania dotyczące infrastruktury były spychane na świeżo zatrudnionych członków zespołów lub trafiały do rzadko spotykanych inżynierów, którzy wykazywali duże zainteresowanie obszarem DevOps. W najlepszym wypadku prowadziło to do sytuacji, gdy jedna lub dwie osoby stawały się ekspertami w dziedzinie infrastruktury i samodzielnie pisały skrypty Terraformu dla całego zespołu. Jednakże takie osoby zwykle nie zagrzewały długo miejsca w zespołach aplikacyjnych, więc zadania znowu trafiały do świeżo zatrudnionych osób, które zazwyczaj wprowadzały spory bałagan w infrastrukturze.
2. Oprócz opisanych powyżej problemów sytuację pogorszał fakt, że po całym przedsiębiorstwie były rozsiane osoby, które mniej lub bardziej zręcznie dłubały skrypty i skrypciki Terraformu. Typową reakcją kierownictwa na takie trudności była centralizacja kompetencji infrastrukturalnych (dla wybranych zespołów lub wręcz dla całego przedsiębiorstwa). Jednakże centralizacja ta zwykle nie miała na celu stworzenia platformy zarządzania infrastrukturą, lecz raczej stworzenie zespołu specjalistów Terraformu oferujących usługi pisania skryptów dla reszty organizacji.
3. Jeżeli jedynym zadaniem dla zcentralizowanych zespołów było pisanie skryptów Terraformu na zlecenie innych jednostek organizacyjnych, to bardzo łatwo było zacząć postrzegać pracujących tam inżynierów jedynie jako trybiki w linii montażowej, których jedynym sensem istnienia jest wypluwanie powtarzalnych fragmentów kodu. Takie nastawienie odstraszało doświadczonych i zaawansowanych deweloperów (takich, którzy są w stanie zaproponować i wprowadzić w strukturze kodu zmiany skutkujące bardziej ogólnym rozwiązaniem, pracującym na wyższym poziomie abstrakcji). W rezultacie z czasem

jakość wytwarzanego kodu się pogorszała, a jego struktura zaczynała przypominać splątany kłąb spaghetti. Taka sytuacja istotnie spowalniała pracę zespołów aplikacyjnych, których potrzeby były choćby nieco odmienne od standardów. Nie wspominając już o ucieleśnianiu najgorszych koszmarów specjalistów do spraw bezpieczeństwa.

Kluczem do sukcesu jest uświadomienie sobie, że organizacja potrzebuje rozwiązania bardziej ujednoliconego i spójnego, niż może dostarczyć centralny zespół wytwarzania skryptów Terraformu. Trzeba zaplanować ewolucję tej grupy ekspertów z centrum utrzymania kleju do centrum inżynierskiego zdolnego wytworzyć złożony produkt — konkretnie platformę. W tym celu musisz dogłębnie poznać potrzeby swoich klientów, aby zrozumieć, jakie usługi warto im udostępniać, i opracować rozwiązanie wykraczające poza proste udostępnianie elementów infrastruktury — a na pewno nie skupiać się na ułatwianiu każdemu dostępu do wszelkich usług, jakich sobie zażyczą.

W miarę wdrażania nowych modeli zarządzania infrastrukturą należy centralizować wiedzę ekspercką i optymalizować wydajność pracy. Zamiast zatrudniać w każdym zespole niezależnych inżynierów DevOps lub SRE, udostępniaj do zarządzania infrastrukturą konkretnych aplikacji ekspertów z puli zespołu wdrażającego platformę. Pracując wspólnie, będą mogli oni poszerzać swoje kompetencje oraz proponować spójne rozwiązania mające zastosowanie szerzej w organizacji. W ten sposób nie tylko będziesz wspierać pojedyncze zmiany w obrębie aplikacji, ale także wykorzystasz doświadczenie tych inżynierów do stworzenia platform przykrywających złożoność infrastruktury leżącej u podstaw aplikacji. A wtedy zacznie się dziać prawdziwa magia.

Centralizacja kosztów migracji

Będziemy wspominać o migracjach jeszcze wiele razy w tej książce, ponieważ uważamy, że zarządzanie migracjami jest istotną porcją korzyści wynikających ze stosowania platform. Czas życia aplikacji i usług elementarnych jest przeważnie dosyć długi, a ich cykle produkcyjne, podczas których przechodzą liczne zmiany, zazwyczaj są od siebie niezależne. W efekcie często dochodzi do jednoczesnego pojawiania się licznych zmian, których obsłużenie istotnie zwiększa koszty utrzymania. Oto sposoby, w jakie stosowanie inżynierii platform może te koszty obniżyć:

Zmniejszenie różnorodności wykorzystywanych komponentów open source i usług chmurowych

Im mniej usług elementarnych wykorzystujesz, tym mniejsza szansa, że przyjdzie Ci przeprowadzić migrację aplikacji z powodu zmian w jednej z nich.

Hermetyzacja komponentów open source i pochodzących od dostawców komercyjnych za pomocą API

Platforma poprzez swoje API oferuje warstwę abstrakcji, która separuje aplikacje od specyfiki działania poszczególnych komponentów elementarnych. Co prawda ta separacja zwykle nie jest doskonała, ale już „wystarczająco dobre" API pozwala uchronić aplikację przed wieloma zmianami, które byłyby wymuszone w przypadku bezpośredniego wykorzystywania komponentu.

Umożliwienie monitorowania wykorzystania platformy

Platformy mogą udostępniać różnorodne mechanizmy standaryzujące zbieranie metadanych dotyczących wszelkich aspektów użycia platformy i wykorzystywanych komponentów. Taki wgląd w stan zależności aplikacji wykorzystujących platformę może pozwolić na redukcję nakładów niezbędnych do przeprowadzenia aktualizacji, gdy któryś z elementów zależnych wymaga zmiany.

Przekazanie odpowiedzialności za komponenty open source i usługi chmurowe zespołom zatrudniającym programistów

Z czasem każde API może się okazać niewystarczające i wymagać zmiany. W skład zespołów zarządzania platformą wchodzą także programiści, więc, inaczej niż w przypadku tradycyjnych zespołów zarządzania infrastrukturą, możliwe jest wytworzenie nietrywialnych narzędzi sprawiających, że proces migracji przestaje być obciążeniem dla większości zespołów aplikacyjnych.

Umożliwienie deweloperom aplikacji utrzymywania ich produktów

Koncepcja DevOps ma na celu uproszczenie określania odpowiedzialności, zgodnie z zasadą „kto buduje, ten utrzymuje". I choć jest ona popularna już od ponad dekady, to wiele przedsiębiorstw nie było w stanie wdrożyć tego modelu pracy. Uważamy, że jednym z ważniejszych czynników sukcesu w organizacjach, którym się to udało, było wykorzystanie warstw abstrakcji w stosowanych platformach, dzięki czemu ukrywana była złożoność operacyjna wykorzystywanych komponentów.

Dyżury utrzymaniowe nie są ulubionym zadaniem pracowników. Ale zauważyliśmy, że gotowość do podjęcia dyżuru przez członków zespołów zaskakująco wzrasta, gdy wsparcie ma dotyczyć wyłącznie problemów z aplikacją tworzoną przez ten zespół. W sumie dlaczego mieliby nie wspierać operacyjnie działania kluczowych systemów biznesowych, jeżeli wcześniej spędzali całe dnie na ich budowie? Jednakże w zbyt wielu przedsiębiorstwach najbardziej widoczne i dotkliwe są problemy powodowane przez infrastrukturę, komponenty open source oraz kod integrujący — przykrywające problemy z samym kodem aplikacji.

Dobrze widać to na przykładzie aplikacji o wymaganej wysokiej odporności na awarie, które są instalowane w wielu strefach dostępności, wielu regionach chmury lub wielu centrach danych. W ten sposób zespoły aplikacyjne są narażone bezpośrednio na wpływ pojawiających się co jakiś czas problemów u dostawcy usług chmurowych, takich jak brak łączności sieciowej. A to znaczy, że dyżurni niechybnie będą doświadczać alarmów podnoszonych o drugiej w nocy. Wykorzystując inżynierię platform, można poradzić sobie z tą niedogodnością poprzez budowę warstwy abstrakcji dla usług wysokiej odporności. Dzięki temu awaryjne przełączenie aplikacji do zapasowego obszaru przetwarzania może być wykonane przez samą platformę, bez angażowania zespołu aplikacyjnego, co ograniczy nocne telefony budzące dyżurnego.

W momencie gdy wykorzystywane systemy zostaną ukryte za warstwą abstrakcji, zespół dostarczający platformę może przejąć pełną odpowiedzialność operacyjną za te systemy. Bardzo ważne jest ograniczanie liczby opcji wspieranych w ten sposób, dzięki czemu łatwiej definiować warstwy abstrakcji, które z kolei mogą być używane przez bardziej różnorodne aplikacje.

Koniecznie musisz także utrzymywać wysokie standardy pracy operacyjnej w obrębie zespołu platformowego, aby deweloperzy aplikacji mogli na nich polegać.

Owszem, trzeba przyznać, że stworzenie i utrzymywanie tak zdefiniowanych platform nie jest łatwe, zwłaszcza iż trzeba przekonać zespoły aplikacyjne do zaakceptowania ograniczonego wyboru opcji. Ale jedyne dostępne alternatywy polegają albo na wystawieniu całej organizacji na bezpośrednią interakcję z elementami infrastruktury (wraz z wszelkimi powiązanymi problemami), albo na ciągłym wykorzystywaniu zespołów operacji i utrzymania (występujących pod dowolnie wybranymi nazwami), co z kolei prowadzi do utrwalenia niejasnej struktury odpowiedzialności, obniżenia wydajności i jakości prac w zwinnych zespołach produktowych oraz ciągłego wzajemnego przerzucania winy w obrębie przedsiębiorstwa.

Zwiększanie autonomii zespołów, aby mogły skupić się na budowaniu platform

Jeżeli chcesz w swojej organizacji wykorzystywać komponenty open source i usługi elementarne dostarczane przez zewnętrznych dostawców, ale jednocześnie chcesz ograniczać negatywny wpływ złożoności rozwiązań na tempo rozwoju, to zdecydowanie musisz powołać zespoły, które stworzą platformy umożliwiające zarządzanie tymi komponentami, wraz z całym dobrodziejstwem inwentarza. Popularne są obecnie cztery metody organizacji pracy powiązane z tematem wdrażania platform, ale żadna z nich nie łączy odpowiedniego zestawu umiejętności i podejścia do pracy wymaganych podczas tworzenia platform. W tabeli 1.1 podsumowaliśmy najważniejsze cechy tych metod oraz wskazaliśmy, dlaczego nie nadają się one do realizacji platform.

Tabela 1.1. Metody organizacji pracy powiązane z platformami, ale niewystarczające do tworzenia platform

Metoda	Priorytet metody	Dlaczego metoda nie jest wystarczająca do tworzenia platform?
Bezpośrednie zarządzanie infrastrukturą	Stabilne funkcjonowanie wykorzystywanej infrastruktury	Niewielkie zainteresowanie tworzeniem warstw abstrakcji infrastruktury wspomagających upraszczanie aplikacji, zwłaszcza gdy aplikacje korzystają z wielu elementów infrastruktury
DevTools	Wydajność pracy programistów, ale tylko do momentu przekazania produktu do środowiska produkcyjnego	Niewielkie zainteresowanie rozwiązywaniem problemów powiązanych z uruchamianiem systemów w środowiskach produkcyjnych o złożonej infrastrukturze
DevOps	Instalacja aplikacji w środowiskach produkcyjnych	Niewielkie zainteresowanie wytwarzaniem bardziej generycznych i uniwersalnych narzędzi automatyzujących pracę, aby mogły być wykorzystywane przez szerokie grono odbiorców
SRE	Niezawodność systemów	Niewielkie zainteresowanie innymi problemami o charakterze systemowym niż niezawodność, oddziaływanie na przedsiębiorstwo zwykle realizowane przez tworzenie zasad organizacji pracy, a nie przez projektowanie lepszych systemów

Rozumiemy, że konkretne osoby, także pracujące w zespołach opatrzonych jedną z powyższych etykiet, czasem zdecydowanie preferują budowanie platform niż wytwarzanie kleju — niemniej organizacyjnie nie mogą się tym zajmować. W naszej książce skupiamy się na ewolucji stosowania tych metod organizacji pracy w przedsiębiorstwach oraz na typowych sposobach definiowania misji konkretnych zespołów, a nie na opisywaniu nastawienia pojedynczych osób. Jest to bardzo istotny problem, ponieważ działania konkretnych pracowników są ograniczane poprzez cel istnienia ich zespołu, a zamiana definicji tego celu bywa trudna, zwłaszcza gdy pozostała część organizacji oczekuje, że zespół po prostu będzie robił to samo co do tej pory.

Wdrożenie inżynierii platform w przedsiębiorstwie daje inżynierom wymienionych specjalności możliwość wydobycia się z silosu organizacyjnego, do którego organizacja ich przypisała, i rozpoczęcia współpracy w ramach zespołów, których zadania są szerzej postawione i obejmują tworzenie platform wprowadzających równowagę w całej organizacji.

- *Zespoły zarządzania infrastrukturą* powinny skupić się na wprowadzaniu równowagi pomiędzy bogactwem oferowanych opcji infrastruktury a prostotą wspomagającą pracę programistów aplikacji.
- *Zespoły DevTools* powinny skupić się na wprowadzaniu równowagi pomiędzy jakością pracy programistów a jakością pracy administratorów utrzymujących środowisko produkcyjne.
- *Zespoły DevOps* powinny skupić się na wprowadzaniu równowagi pomiędzy kodem integrującym optymalnie dopasowanym do każdej aplikacji oddzielnie a oprogramowaniem o bardziej generycznym zestawie funkcjonalności, które może wspierać wiele różnych aplikacji.
- *Zespoły SRE* powinny skupić się na wprowadzaniu równowagi między niezawodnością systemów a innymi ich atrybutami, takimi jak elastyczność usług, efektywność kosztowa lub wydajność przetwarzania.

Wdrażanie inżynierii platform pozwala całkowicie przedefiniować oczekiwania organizacji i powołać zespół, który faktycznie skupi się na wytwarzaniu technologii prowadzących do oczyszczenia bagna.

Czy stosowanie platform sprzyja innowacjom?

Mamy nadzieję, że zaczynasz dostrzegać, jak stosowanie platform może przeciwdziałać wszelkim bolączkom trapiącym zespoły deweloperskie, poprawiać szybkość działania i poziom bezpieczeństwa systemów, sprawiać, że programiści będą bardziej produktywni, zapewniać automatyczne migracje aplikacji oraz przyspieszać realizację wszelkich zadań poprzez przyspieszoną wymianę informacji zwrotnej. Co prawda mamy świadomość, że osiągnięcie takich rezultatów może zająć całkiem sporo czasu, ale uważamy, iż warto do takiego idealnego stanu dążyć.

Ale co jeszcze dobrego może Cię spotkać, gdy zaczniesz wdrażać platformy? Jesteśmy inżynierami, więc mamy naturalne oczekiwanie, że nasze platformy wspomogą też innowacje i prowadzenie eksperymentów, bo innowacje są motorem rozwoju naszych przedsiębiorstw. I faktycznie, platformy mogą wspierać innowacje. Ale musimy w tym obszarze dać nieco szersze wyjaśnienia, gdyż równie łatwo zastosowanie platform może innowacjom i eksperymentom zaszkodzić.

Platformy zdecydowanie wspierają innowację rozumianą jako innowacja biznesowa realizowana w kontekście istniejących technologii. Platformy pozwalają deweloperom aplikacji szybciej dostarczać kolejne wersje produktów dzięki zwiększeniu ogólnej produktywności, a szczególnie dzięki umożliwieniu im bezpiecznego instalowania nowych funkcjonalności w środowiskach produkcyjnych (na przykład za pomocą konfiguracyjnych przełączników udostępniania funkcji oraz testów A/B). W ten sposób możliwe jest szybkie weryfikowanie pomysłów biznesowych opartych na istniejących technologiach.

Jednakże są także rodzaje innowacji, których platformy ze swojej natury nie tylko nie wspierają, ale wręcz utrudniają ich realizowanie. Większość innowacyjnych pomysłów technologicznych może wymagać użycia narzędzi, które kompletnie nie istnieją w ramach organizacji. Świetnym przykładem jest obszar przetwarzania danych, ponieważ rozwój tej dziedziny jest błyskawiczny. Możesz mieć doskonałą platformę, która zapewnia łatwe wykorzystanie relacyjnych baz danych, dzięki czemu większość inżynierów w Twoim przedsiębiorstwie może łatwo realizować swoje zadania. Wyobraź sobie, że jeden z zespołów pracuje nad innowacyjnym rozwiązaniem dającym ogromne szanse na rozwój biznesowy, a relacyjne bazy danych oferowane przez platformę nie będą spełniały specyficznych wymagań wydajnościowych tego rozwiązania. W takim wypadku zespół porzuci korzystanie z platformy, przynajmniej częściowo, aby zbudować swoją aplikację. Jeżeli w przyszłości okaże się, że nowe rozwiązanie przynosi wymierne korzyści, to możesz zdecydować o włączeniu nowego systemu przechowywania danych do oferty dostępnej poprzez platformę. Ale ten rodzaj innowacji nie był umożliwiony przez istnienie platformy. Przy czym nie należy też przesadnie ochoczo wciskać na platformę każdego nowego pomysłu, jaki się wykluje w przedsiębiorstwie. Raczej należy pozwolić takim pomysłom na niezależny rozwój, a do oferty platformy dołączać wyłącznie udane, na które pojawia się coraz szerszy popyt.

Zespoły utrzymujące platformy doświadczają silnej pokusy, aby w zarodku dusić wszelkie innowacyjne inicjatywy powodujące odpływ użytkowników z platformy. W większości przypadków wcielanie w życie takich pomysłów faktycznie jest stratą sił i środków, gdyż próba stworzenia nowej implementacji wynika wyłącznie ze spotykanego wśród inżynierów na całym świecie błędu poznawczego skutkującego przekonaniem, że tylko samodzielnie stworzone rozwiązania problemów są dobre, a wszystko inne „od złego pochodzi". Ale zdarzają się też przypadki słusznego poszukiwania rozwiązań poza dostępnym standardem. Zespół platformowy może łatwo zabić wszelką zdrową innowacyjność w firmie albo doprowadzić do tego, że oferta funkcjonalności na platformie jest zbyt szeroka, by przedsiębiorstwo czerpało odpowiednio duże korzyści z istnienia platformy. Wystarczy, żeby zespół platformowy nigdy nie pozwalał na żadne odstępstwa w stosowaniu funkcjonalności platformy albo domagał się, by wszelkie nowe rozwiązania mogły być wdrażane wyłącznie przez ten zespół.

Podsumowując: platformy powinny ułatwić proste eksperymenty i działania innowacyjne wykonywane w zakresie znanych i używanych technologii, gdyż platformy sprawiają, że deweloperzy są bardziej produktywni i skupieni na warstwie aplikacji. Ale zespół utrzymujący platformę nie będzie wspierał innowacji w dowolnym zakresie. W praktyce, aby umożliwić istotne innowacyjne zmiany, musisz pozwolić niektórym zespołom na realizację i wykazanie wartości własnych nowych rozwiązań. Jedną z kluczowych umiejętności lidera inżynierii platform jest umiejętność rozpoznania, kiedy warto przymusić zespoły do korzystania z rozwiązań centralnych, a kiedy można pozwolić na stworzenie alternatywnych nieautoryzowanych platform[9]. Będziemy szerzej omawiać tę umiejętność w rozdziale 10.

Podsumowanie

Naszą branżę czeka twarde zderzenie z paraliżem wynikającym z rosnącej złożoności rozwiązań — niektórzy już dotykają tej ściany. Musimy znaleźć rozwiązanie, niezależnie od tego, czy paraliż grozi nam z powodu coraz mniej wydajnych zespołów DevOps, które są zasypywane lawiną konsekwencji milionów drobnych decyzji związanych z zarządzaniem infrastrukturą w postaci kodu, czy z powodu kaskadowo nawarstwiających się aktualizacji i migracji komponentów oprogramowania powiązanych siecią zależności. Wierzymy, że inżynieria platform jest tym rozwiązaniem i jej znaczenie dla branży technologicznej wciąż rośnie. Łącząc w jednym zespole nastawienie na tworzenie produktów z doświadczeniem eksperckim inżynierów oprogramowania i inżynierów systemów, możemy okiełznać złożoność rozwiązań paraliżującą przedsiębiorstwo.

[9] W świecie platform jest to ekwiwalent nieautoryzowanych systemów IT (ang. *shadow IT*), czyli systemów wdrażanych niezależnie w różnych działach z pominięciem centralnych struktur organizacyjnych odpowiedzialnych za dostarczanie systemów. Taki sposób wdrożenia ma na celu wypełnienie luk w ofertach systemów centralnych lub ominięcie ograniczeń nakładanych przez systemy centralne.

Platforma to organizm, a jego sercem jest zespół!

Czy radzisz sobie z zarządzaniem kodem, narzędziami i infrastrukturą użytkowaną przez wiele zespołów? Próby rozwiązania tego problemu trwają od ćwierćwiecza, nierzadko jednak ignoruje się potrzeby użytkowników i nie zapewnia wymaganej niezawodności. Efekt? Totalny chaos i przytłaczające złożonością systemy. A przecież można inaczej: oto inżynieria platform!

Ten praktyczny przewodnik docenią głównie inżynierowie, menedżerzy, menedżerzy produktu i liderzy. Jego treść wykracza poza aspekty programistyczne, prezentując zalety całościowego myślenia o infrastrukturze, a więc szerszego planowania i uwzględniania kwestii administracji systemami. Dzięki książce poznasz odpowiedzi na kluczowe pytania: kiedy powołać projekt wytwarzania platformy, jak zorganizować zespół zajmujący się platformą, co odróżnia planowanie platformy od innych, pozornie podobnych inicjatyw. Znajdziesz tu też przydatne wskazówki, z których możesz skorzystać na każdym etapie wdrażania i stosowania podejścia opartego na inżynierii platform.

Najważniejsze zagadnienia:

- platforma jako produkt dla deweloperów
- rola i granice działania zespołów inżynierii platform
- wprowadzanie inżynierii platform w organizacji
- wyzwania związane ze skalowaniem platform
- najlepsze praktyki lidera zespołów inżynierii platform

Inżynieria platform to sport zespołowy. A to jest Twój plan gry!

Kelsey Hightower, współautor książki *Kubernetes. Tworzenie niezawodnych systemów rozproszonych*

Camille Fournier specjalizuje się w prowadzeniu zespołów w różnych organizacjach, w tym z listy Fortune 50. Jest założycielką komitetu nadzoru technicznego Cloud Native Computing Foundation i autorką książki *Od inżyniera do menedżera.*

Ian Nowland ma duże doświadczenie w branży informatycznej. Od 2008 do 2016 roku pracował w AWS, gdzie prowadził projekty Amazon EMR i EC2 Nitro. Obecnie zajmuje się rozwijaniem startupu, którego jest współzałożycielem.

Helion
helion.pl
HELION S.A.
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-289-2555-7
9 788328 925557
Cena: 89,00 zł