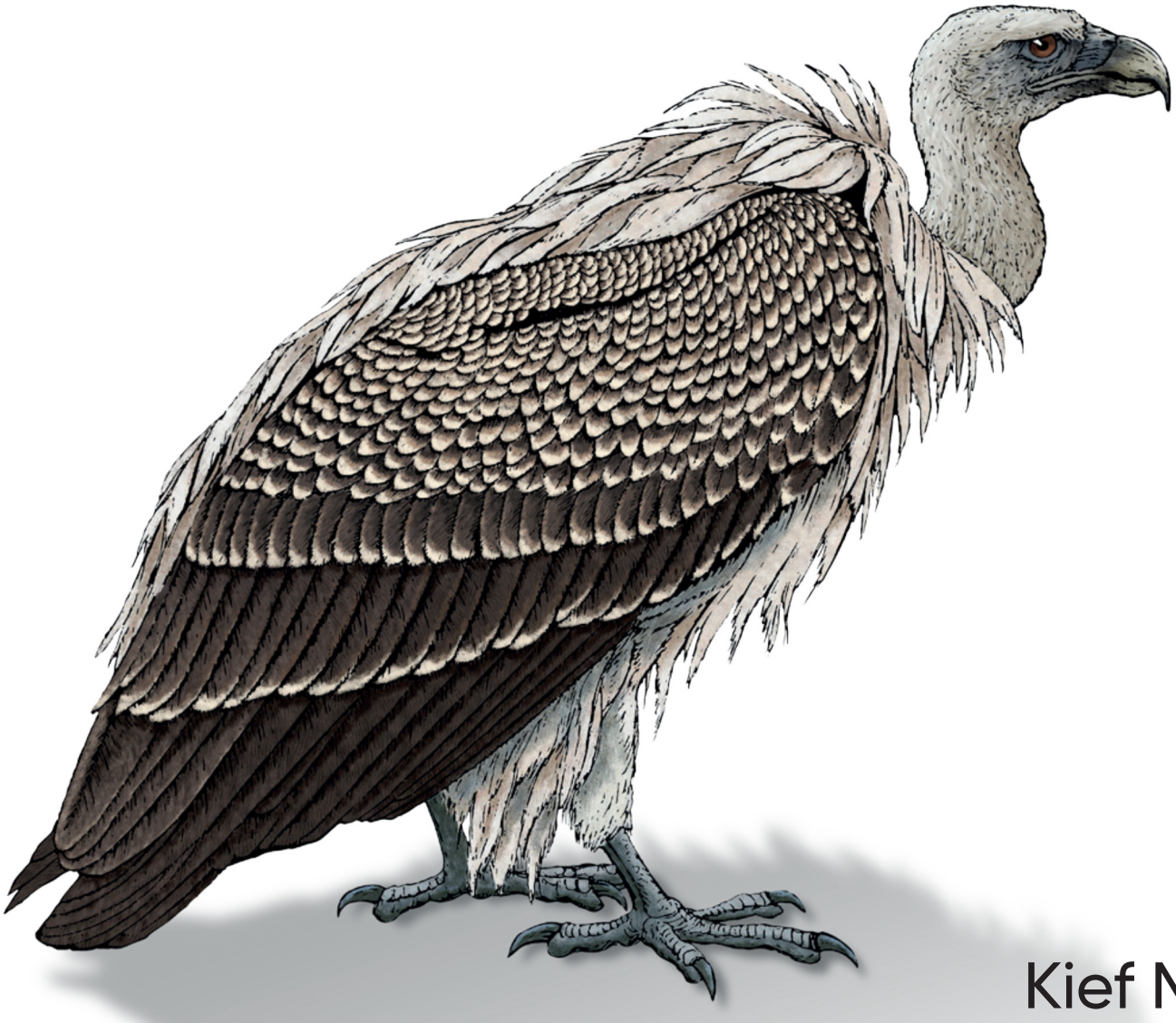
O'REILLY®

Infrastruktura jako kod

Dynamiczne systemy w epoce chmury



Kief Morris

Infrastruktura jako kod

Dynamiczne systemy w epoce chmury

Kief Morris

przekład: Janusz Machowski

APN Promise
Warszawa 2021

O'REILLY®

Infrastruktura jako kod

Authorized translation of English edition of
Infrastructure as Code, Second Edition
ISBN 978-1-098-11467-1

APN PROMISE SA, ul. Domaniewska 44a, 02-672 Warszawa
tel. +48 22 35 51 600, fax +48 22 35 51 699
e-mail: mspress@promise.pl

ISBN: 978-83-7541-442-4 (wyd. drukowane), 978-83-7541-446-2 (ebook)

Projekt okładki: Karen Montgomery
Ilustracje: John Francis Amalanathan
Ilustracja na okładce: Jose Marzan

Przekład: Janusz Machowski
Redakcja: Marek Włodarz
Korekta: Ewa Swędrowska
Skład i łamanie: MAWart Marek Włodarz

Spis treści

Część III. Praca z serwerami i innymi platformami wykonawczymi aplikacji

Przedmowa

Dziesięć lat temu dyrektor ds. informatyki w globalnym banku wyśmiał mnie, gdy zaproponowałem, aby przyjrzano się technologiom chmur prywatnych i narzędziom do automatyzacji infrastruktury: „Takie rzeczy mogą być dobre dla start-upów, a my jesteśmy zbyt duzi i mamy zbyt skomplikowane wymagania". Jeszcze kilka lat temu wiele przedsiębiorstw uważało, że korzystanie z chmur publicznych jest wykluczone.

Dzisiaj technologia chmury jest wszechobecna. Nawet największe, najbardziej zatwardziałe organizacje błyskawicznie przyjmują strategię „najpierw chmura". Te organizacje, które uznają za niemożliwe korzystanie z chmur publicznych, wdrażają w swoich centrach danych dynamicznie udostępniane platformy infrastrukturalne[1]. Możliwości tych platform ewoluują i są ulepszane tak, że trudno je ignorować nie ryzykując pozostania w tyle za innymi.

Chmura i technologie automatyzacji usuwają bariery utrudniające wprowadzanie zmian w systemach produkcyjnych, a to stawia nowe wyzwania. Choć większość organizacji chce przyśpieszyć tempo zmian, nie mogą one pozwolić sobie na ignorowanie ryzyka i potrzeby nadzoru. Tradycyjne procesy i techniki bezpiecznej zmiany infrastruktury nie są przystosowane do szybkiego wprowadzania zmian. Tego typu metody pracy ograniczają zwykle korzyści płynące z nowoczesnych technologii „epoki chmury" – spowalniają pracę i szkodzą stabilności[2].

W rozdziale 1 używam terminów „epoka żelaza" i „epoka chmury" („Od epoki żelaza do epoki chmury" na stronie 4), aby opisać różne filozofie zarządzania infrastrukturą fizyczną, w których poprawianie błędów jest wolne i kosztowne, oraz zarządzanie infrastrukturą wirtualną, w której wykrywanie i naprawa błędów odbywa się szybko.

Narzędzia infrastruktury jako kodu dają możliwość pracy w sposób, który pomaga wprowadzać zmiany częściej, szybciej i skuteczniej, poprawiając ogólną jakość systemów.

1 Na przykład wiele organizacji rządowych i finansowych w krajach, w których nie ma chmury, ma prawny zakaz hostingu danych i transakcji zagranicą.

2 Z badań opublikowanych przez DORA w *State of DevOps Report* (*https://oreil.ly/ysk9n*) wynika, że rozbudowane procesy zarządzania zmianami cechuje słaba jakość pod względem niepowodzenia zmian i innych mierników efektywności dostarczania oprogramowania.

Ale korzyści nie są efektem samych narzędzi. Są wynikiem sposobu używania tych narzędzi. Sztuka polega na tym, aby wykorzystać technologię do wbudowania jakości, niezawodności i zgodności w proces dokonywania zmian.

Dlaczego napisałem tę książkę

Pierwsze wydanie tej książki napisałem dlatego, że nie widziałem nigdzie spójnego zbioru wskazówek, jak zarządzać infrastrukturą jako kodem. Było mnóstwo porad porozrzucanych po blogach, dyskusjach konferencyjnych oraz dokumentacjach różnych produktów i projektów. Ale zwykły praktyk musiał przejrzeć to wszystko, aby ułożyć z tego strategię dla siebie, a większość ludzi po prostu nie ma na to czasu.

Wrażenia z pisania pierwszego wydania były niesamowite. Była to dla mnie okazja do podróży i rozmów z ludźmi na całym świecie o ich własnych doświadczeniach. Rozmowy te dały mi nowe spojrzenie i postawiły przede mną nowe wyzwania. Dowiedziałem się, że pisanie książki, przemawianie na konferencjach i konsultowanie się z klientami sprzyja rozmowom. Jako branża wciąż gromadzimy, udostępniamy i rozwijamy nasze pomysły zarządzania infrastrukturą jako kodem.

Co dodałem i co zmieniłem w tym wydaniu

Zaszły zmiany od pierwszego wydania tej książki w czerwcu 2016. Tamto wydanie miało podtytuł „Zarządzanie serwerami w chmurze", co odzwierciedlało fakt, że większość automatyzacji infrastruktury do tamtego momentu była skoncentrowana na konfigurowaniu serwerów. Od tamtej pory kontenery i klastry stały się znacznie ważniejsze, a działania dotyczące infrastruktury przeniosły się na zarządzanie kolekcjami zasobów infrastruktury, udostępnianymi przez platformy chmurowe – które nazywam w tej książce *stosami*.

W efekcie to nowe wydanie zawiera szersze omówienie tworzenia stosów, co jest zadaniem narzędzi takich, jak CloudFormation i Terraform. Przyjąłem pogląd, że wykorzystujemy narzędzia zarządzania stosami do tworzenia kolekcji infrastruktur zapewniających środowisko wykonawcze dla aplikacji. Te środowiska wykonawcze mogą obejmować serwery, klastry i bezserwerowe środowiska wykonawcze.

Zmieniłem sporo na podstawie tego, czego się dowiedziałem o zmieniających się wyzwaniach i potrzebach zespołów tworzących infrastrukturę. Jak już wspomniałem w tej przedmowie, kluczową zaletą infrastruktury jako kodu jest według mnie pokazanie bezpiecznej i łatwej zmiany infrastruktury. Uważam, że ludzie nie doceniają wagi tego, myśląc, że infrastruktura to coś, co się tworzy i potem zapomina o tym.

Ale zbyt wiele zespołów, które spotykam, walczy o zaspokojenie potrzeb swoich organizacji; nie potrafią zapewnić wystarczająco szybkiego rozwoju i skalowania, dostarczania oprogramowania ani oczekiwanej niezawodności i bezpieczeństwa. A kiedy zagłębiamy się w szczegóły zadań, okazuje się, że ludzie są przytłoczeni potrzebą aktualizacji, naprawiania i ulepszania swoich systemów. Dlatego poświęciłem temu dwa razy więcej miejsca, czyniąc głównym tematem książki.

W tym wydaniu przedstawione są trzy podstawowe praktyki używania infrastruktury jako kodu, dzięki którym zmiany powinny być łatwe i bezpieczne:

Definiuj wszystko jako kod
: To wynika oczywiście z tytułu i pozwala uzyskać powtarzalność i spójność.

Nieustannie testuj i dostarczaj sukcesywnie
: Każda zmiana zwiększa bezpieczeństwo. Pozwala również iść naprzód szybciej i z większą pewnością.

Twórz małe, proste elementy, które można zmieniać niezależnie od innych
: Zmienianie ich jest łatwiejsze i bezpieczniejsze niż dużych elementów.

Te trzy praktyki wzajemnie się wzmacniają. Kod jest łatwy do śledzenia, wersjonowania i dostarczania między różnymi etapami procesu zarządzania zmianą. Łatwiej jest nieustannie testować mniejsze elementy. Ciągłe, niezależne testowanie każdego elementu zmusza do zachowania luźno sprzężonego projektu.

Te praktyki i szczegóły ich stosowania są znane ze świata tworzenia oprogramowania. W pierwszym wydaniu tej książki przybliżyłem praktyki tworzenia i dostarczania oprogramowania zwinnego. W tym wydaniu przybliżam dodatkowo reguły i praktyki projektowania efektywnego.

W ciągu ostatnich kilku lat widziałem zespoły borykające się z dużymi i skomplikowanymi systemami infrastruktury i dostrzegłem korzyści płynące ze stosowania wiedzy opartej na wzorcach i regułach projektowania oprogramowania, dlatego zamieściłem w tej książce kilka rozdziałów na ten temat.

Zauważyłem też, że organizacja i praca nad kodem infrastruktury stanowi dla wielu zespołów problem, dlatego omówiłem różne bolączki. Opisałem, jak można dobrze organizować bazy kodu, jak zapewniać instancje programowania i testów dla infrastruktury i jak zarządzać współpracą wielu osób, łącznie z tymi, które zajmują stanowiska kierownicze.

Co dalej

Nie sądzę, abyśmy jako branża osiągnęli dojrzałość w zarządzaniu infrastrukturą. Mam nadzieję, że ta książka daje przyzwoity obraz tego, jakie zespoły są dzisiaj skuteczne. I odrobinę aspiracji, co możemy zrobić lepiej.

Jestem przekonany, że w ciągu następnych pięciu lat łańcuchy narzędzi i podejścia będą ewoluować. Możemy zobaczyć więcej języków ogólnego przeznaczenia używanych do tworzenia bibliotek i możemy dynamicznie generować infrastrukturę, zamiast definiować statyczne szczegóły środowisk na niskim poziomie. Z pewnością musimy lepiej zarządzać zmianami w działającej już infrastrukturze. Większość znanych mi zespołów boi się stosować kod do działającej infrastruktury. (Jeden z zespołów określił Terraform jako „Terrorform," ale użytkownicy innych narzędzi też tak czują).

Czym jest i czym nie jest ta książka

Teza tej książki jest taka, że badanie różnych sposobów wykorzystywania narzędzi do implementowania infrastruktury może pomóc nam poprawić jakość świadczonych usług. Naszym celem jest wykorzystanie szybkości i częstotliwości dostaw do poprawy niezawodności i jakości tego, co dostarczamy.

Dlatego książka mniej się skupia na konkretnych narzędziach, a bardziej na sposobie ich używania.

Chociaż wymieniam przykłady narzędzi do wykonywania konkretnych funkcji, jak konfigurowanie serwerów i udostępnianie stosów, nie ma tu szczegółowych informacji o sposobie korzystania z konkretnych narzędzi albo platform chmurowych. Są natomiast wzorce, praktyki i techniki odpowiednie dla wszystkich narzędzi i platform.

Nie ma w książce przykładów kodu ze świata rzeczywistych narzędzi i chmur. Narzędzia zmieniają się zbyt szybko w tej dziedzinie, aby przykładowy kod zachował aktualność. Natomiast zawarte porady powinny się starzeć wolniej i pasować do różnych narzędzi. Dla zilustrowania koncepcji w przykładach posługuję się pseudokodem i fikcyjnymi narzędziami. Odniesienia do przykładowych projektów i kodu można znaleźć na związanej z książką stronie *https://infrastructure-as-code.com*.

Ta książka nie zawiera wskazówek na temat używania systemu operacyjnego Linux ani konfigurowania klastra Kubernetes czy routingu sieciowego. Zakres książki obejmuje natomiast sposoby udostępniania zasobów infrastruktury w celu utworzenia tych elementów oraz sposoby wykorzystania kodu do ich dostarczania. Pokazuję różne wzorce topologii klastra i podejścia do definiowania klastrów i zarządzania nimi jak kodem. Opisuję wzorce udostępniania, konfigurowania i zmieniania instancji serwerów za pomocą kodu.

Praktyki przedstawione w tej książce należy uzupełnić zasobami konkretnych systemów operacyjnych, technologii klastrów i platform chmurowych. Podkreślam raz jeszcze, ta książka wyjaśnia podejście do wykorzystywania narzędzi i technologii bez względu na to, jakie to są narzędzia.

Książka traktuje również lekko tematy związane z obsługą, takie jak monitorowanie i obserwowanie, agregację dzienników, zarządzanie tożsamościami i inne elementy potrzebne do wspierania usług w środowisku chmurowym. To, co w niej jest, powinno pomóc w zarządzaniu infrastrukturą potrzebną dla tych usług jak kodem, ale szczegóły konkretnych usług są znowu czymś, co można znaleźć w bardziej konkretnych źródłach.

Nieco historii infrastruktury jako kodu

Narzędzia i praktyki infrastruktury jako kodu pojawiły się na długo przed tym terminem. Administratorzy systemów od samego początku używali skryptów do zarządzania systemami. Mark Burgess stworzył pionierski system CFEngine[3] w 1993 roku. Po raz pierwszy nauczyłem się praktyk używania kodu do pełnej automatyzacji udostępniania i aktualizacji serwerów z witryny Infrastructures.org na początku lat 2000[4].

Infrastruktura jako kod rozwijała się wraz z ruchem DevOps. Andrew ClayShafer i Patrick Debois zapoczątkowali ruch DevOps podczas przemówienia na konferencji Agile 2008[5]. Pierwsze przypadki użycia terminu Infrastruktura jako kod pochodzą z wykładu zatytułowanego „Agile Infrastructure"[6], wygłoszonego przez Claya-Shafera na konferencji Velocity w 2009 roku oraz artykułu John Willisa[7] podsumowującego ten wykład. Adam Jacob, współzałożyciel firmy Chef, i Luke Kanies, założyciel Puppet, również używali tego zwrotu w tym czasie.

Dla kogo jest ta książka

Książka jest dla osób związanych z udostępnianiem i wykorzystywaniem infrastruktury przeznaczonej do dostarczania i uruchamiania oprogramowania. Czytelnik może mieć doświadczenie w zakresie systemów i infrastruktury lub w tworzeniu i dostarczaniu oprogramowania. Jego dziedziną może być inżynieria, testowanie, architektura albo zarządzanie. Zakładam pewne doświadczenie z chmurą lub infrastrukturą zwirtualizowaną i narzędziami do automatyzacji infrastruktury przy użyciu kodu.

Czytelnicy, dla których infrastruktura jako kod jest czymś nowym, powinni uznać tę książkę za dobre wprowadzenie do tematu, chociaż najwięcej korzyści przyniesie ona osobom znającym działanie platform chmurowych infrastruktury i podstawy co najmniej jednego narzędzia kodowania infrastruktury.

Ci, którzy mają większe doświadczenie z tymi narzędziami, znajdą tu mieszankę znanych oraz nowych koncepcji i podejść. Staram się stworzyć wspólny język oraz przedstawić wyzwania i rozwiązania w taki sposób, który doświadczeni praktycy i zespoły uznają za przydatny.

3 *https://cfengine.com*

4 Oryginalna treść nadal była dostępna na tej stronie w lecie 2020 i nie była aktualizowana od 2007 r. (*http://www.infrastructures.org*).

5 *https://oreil.ly/ermR3*

6 *https://oreil.ly/qnJKX*

7 *https://oreil.ly/2F6y_*

Zasady, praktyki i wzorce

Stosuję terminy *zasady*, *praktyki* i *wzorce* (i *antywzorce*) na określenie podstawowych pojęć. Oto znaczenie każdego z tych terminów:

Zasada
: Zasada to reguła, która pomaga wybierać między potencjalnymi rozwiązaniami.

Praktyka
: Praktyka to sposób na wdrożenie czegoś. Podana praktyka nie zawsze jest jedynym sposobem na zrobienie czegoś, a nawet może nie być najlepszym sposobem zrobienia tego w konkretnej sytuacji. Należy korzystać z zasad w celu wybrania najbardziej odpowiedniej praktyki w danej sytuacji.

Wzorzec
: Wzorzec jest potencjalnym rozwiązaniem problemu. Jest bardzo podobny do praktyki pod tym względem, że różne wzorce mogą być bardziej skuteczne w różnych sytuacjach. Każdy wzorzec jest opisany w formacie, który powinien pomóc w ocenie, na ile jest on stosowny w przypadku danego problemu.

Antywzorzec
: Antywzorzec jest potencjalnym rozwiązaniem, którego należy unikać w większości sytuacji. Zwykle jest to coś, co wydaje się dobrym pomysłem lub coś, co zaczynamy realizować nie zdając sobie sprawy z konsekwencji.

Dlaczego nie używam terminu „Najlepsza praktyka"

Ludzie z naszej branży uwielbiają mówić o „najlepszych praktykach". Problem z tym terminem jest taki, że wywołuje on często przekonanie, że istnieje tylko jedno rozwiązanie problemu, bez względu na kontekst.

Wolę opisywać praktyki oraz wzorce i zwracać uwagę, kiedy są przydatne i jakie mają ograniczenia. Niektóre z nich opisuję jako skuteczniejsze lub bardziej odpowiednie, ale staram się być otwarty na alternatywy. W przypadku praktyk, które uważam za mniej skuteczne, mam nadzieję, że wyjaśniam, skąd się bierze to przekonanie.

Przykłady z ShopSpinner

Do zilustrowania koncepcji zawartych w tej książce posługuję się fikcyjną firmą o nazwie ShopSpinner. ShopSpinner tworzy i uruchamia sklepy internetowe dla swoich klientów.

ShopSpinner wykorzystuje do działania FCS, Fictional Cloud Service (fikcyjną usługę chmurową), publicznego dostawcę IaaS z usługami obejmującymi FSI (Fictional

Server Images) i FKS (Fictional Kubernetes Service). Do definiowania infrastruktury w chmurze i zarządzania nią ShopSpinner używa narzędzia *Stackmaker* – analogicznego do Terraform, CloudFormation i Pulumi. Serwery są konfigurowane za pomocą narzędzia *Servermaker*, podobnego do Ansible, Chef czy Puppet.

Infrastruktura i projekt systemu ShopSpinner mogą się różnić w zależności od tego, do czego są mi potrzebne, podobnie jak składnia kodu i argumenty wiersza polecenia dla jego fikcyjnych narzędzi.

Konwencje stosowane w tej książce

W książce tej stosowane są następujące konwencje typograficzne:

Kursywa
: Wskazuje nowe terminy, adresy URL, adresy email, nazwy i rozszerzenia plików.

`Czcionka o stałej szerokości`
: Jest używana w listingach programów, a także w akapitach w przypadku odniesień do elementów programu, takich jak nazwy zmiennych czy funkcji, bazy danych, typy danych, zmienne środowiskowe, instrukcje i słowa kluczowe.

`Czcionka o stałej szerokości pogrubiona`
: Wskazuje polecenia lub inne porcje tekstu, które należy wpisać dosłownie.

`Kursywa o stałej szerokości czcionki`
: Wskazuje tekst, który należy zastąpić wartościami podanymi przez użytkownika lub wartościami wynikającymi z kontekstu.

Ten element oznacza wskazówkę lub sugestię.

Ten element oznacza ogólną uwagę.

Ten element oznacza ostrzeżenie lub przestrogę.

Nauka online od O'Reilly

Od ponad 40 lat firma *O'Reilly Media* udostępnia szkolenia, wiedzę i informacje w zakresie technologii i biznesu, aby pomóc firmom w odniesieniu sukcesu.

Nasza wyjątkowa sieć ekspertów i innowatorów dzieli się swoją wiedzą i doświadczeniem za pośrednictwem książek, artykułów i naszej platformy nauczania online. Platforma nauczania online O'Reilly oferuje dostęp na żądanie do kursów szkoleniowych na żywo, pogłębionych ścieżek nauczania, interaktywnych środowisk kodowania oraz ogromnej kolekcji tekstów i filmów pochodzących od O'Reilly i ponad 200 innych wydawców. Więcej informacji można znaleźć na stronie *http://oreilly.com*.

Jak się z nami skontaktować

Komentarze i pytania dotyczące tej książki prosimy kierować do wydawcy:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (USA lub Kanada)
707-829-0515 (połączenia międzynarodowe lub lokalne)
707-829-0104 (faks)

Książka ma swoją stronę internetową z erratą, przykładami i dodatkowymi informacjami. Jej adres to *https://oreil.ly/infra-as-code-2e*.

Komentarze i pytania techniczne dotyczące książki można wysyłać mailem na adres *bookquestions@oreilly.com*.

Wiadomości i informacje o naszych książkach i kursach są na stronie *http://oreilly.com*.

Znajdź nas na Facebooku: *http://facebook.com/oreilly*

Śledź nas na Twitterze: *http://twitter.com/oreillymedia*

Oglądaj nas na YouTube: *http://www.youtube.com/oreillymedia*

Podziękowania

Podobnie jak w przypadku pierwszego wydania, książka ta nie jest wyłącznie moim dziełem. Jest to efekt zestawienia i konsolidacji, najlepiej jak umiem, tego, czego się nauczyłem od innych osób – tak wielu, że nie jestem w stanie ich zapamiętać i prawidłowo skojarzyć. Przeprosiny i podziękowania dla tych wszystkich, których zapomniałem tutaj wymienić.

Zawsze lubię szukać pomysłów z Jamesem Lewisem; nasze rozmowy oraz jego teksty i przemówienia miały bezpośredni i pośredni wpływ na dużą część tej książki. Życzliwie dzielił się swoim doświadczeniem w projektowaniu oprogramowania i szeroką znajomością wielu innych tematów i wyraził opinię na temat niemal skończonej wersji tej książki. Jego sugestie pomogły uściślić powiązania, które próbowałem nakreślić między inżynierią oprogramowania a infrastrukturą jako kodem.

Martin Fowler od samego początku życzliwie wspierał moje wysiłki. Zawsze inspirowała mnie jego umiejętność czerpania z doświadczeń innych ludzi, wykorzystywania swoje wiedzy i spostrzeżeń oraz przekształcania tego wszystkiego w jasne, pomocne rady.

Thierry de Pauw był bardzo rozważnym i pomocnym recenzentem. Przeczytał wiele wersji roboczych i podzielił się swoimi refleksjami, wskazując mi, co uznał za nowe i przydatne, które pomysły pokrywały się z jego własnymi doświadczeniami i które fragmenty nie były dla niego zrozumiałe.

Muszę podziękować Abigail Bangser, Jonowi Barberowi, Maksowi Griffithsowi, Anne Simmons i Claire Walkley za ich zachętę i inspirację.

Osoby, z którymi pracowałem, przekazały mi swoje uwagi i pomysły, które poprawiły jakość książki. James Green podzielił się spostrzeżeniami na temat inżynierii danych i nauczania maszynowego w kontekście infrastruktury. Pat Downey wyjaśnił swój sposób wykorzystania rozszerzania się i kurczenia w odniesieniu do infrastruktury. Vincenzo Fabrizi zwrócił mi uwagę na znaczenie inwersji sterowania dla zależności infrastruktury. Effy Elden to niewyczerpane źródło wiedzy o oprzyrządowaniu infrastruktury. Moritz Heiber bezpośrednio i pośrednio wpłynął na zawartość tej książki, choć nie mam zbyt wielkiej nadziei, że zgadza się z nią w 100%.

W ramach ThoughtWorks miałem okazję do kontaktów i omawiania Infrastruktury jako kodu i tematów pokrewnych z wieloma kolegami i klientami podczas warsztatów,

projektów i na forach internetowych. Niektóre z tych osób to: Ama Asare, Nilakhya Chatterjee, Audrey Conceicao, Patrick Dale, Dhaval Doshi, Filip Fafara, Adam Fahie, John Feminella, Mario Fernandez, Louise Franklin, Heiko Gerin, Jarrad „Barry" Goodwin, Emily Gorcenski, James Gregory, Col Harris, Prince M Jain, Andrew Jones, Aiko Klostermann, Charles Korn, Vishwas Kumar, Punit Lad, Suya Liu, Tom Clement Oketch, Gerald Schmidt, Boss Supanat Pothivarakorn, Rodrigo Rech, Florian Sellmayr, Vladimir Sneblic, Isha Soni, Widyasari Stella, Paul Valla, Srikanth Venugopalan, Ankit Wal, Paul Yeoh i Jiayu Yi. Dziękuję również Kentowi Spillnerowi – tym razem pamiętam dlaczego.

Mnóstwo osób przejrzało różne wersje robocze tego wydania książki i podzieliło się swoimi uwagami, w tym Artashes Arabajyan, Albert Attard, Simon Bisson, Phillip Campbell, Mario Cecchi, Carlos Conde, Bamdad Dashtban, Marc Hofer, Willem van Ketwich, Barry O'Reilly, Rob Park, Robert Quinlivan, Wasin Watthanasrisong i Rebecca Wirfs-Brock.

Wielkie podziękowania dla Virginii Wilson, mojej redaktorki, która pomogła mi przejść przez długi i wyczerpujący proces powstawania tej książki. Mój kolega, John Amalanathan, z wielką cierpliwością i starannością zamienił moje mizerne diagramy w zgrabną grafikę, którą można tutaj oglądać.

Mój pracodawca, firma ThoughtWorks, stanowiła ogromne wsparcie. Po pierwsze, tworząc dla mnie środowisko, w którym mogłem się uczyć od fenomenalnych ludzi, po drugie, wspierając kulturę, która zachęca jej członków do dzielenia się pomysłami z branżą, a po trzecie, wspierając mnie podczas pracy z innymi osobami z ThoughtWorks oraz klientami w celu odkrywania i testowania nowych sposobów pracy. Ashok Subramanian, Ruth Harrison, Renee Hawkins, Ken Mugrage, Rebecca Parsons i Gayathri Rao, między innymi, pomogli mi uczynić z tego coś więcej niż osobisty projekt.

Na koniec i najbardziej dziękuję moim ukochanym, Ozlem i Erelowi, którzy wytrzymali moją obsesję na punkcie tej książki. Kolejny raz.

CZĘŚĆ I

Podstawy

Rozdział 1

Co to znaczy infrastruktura jako kod?

Jeśli ktoś pracuje w zespole tworzącym i obsługującym infrastrukturę IT, to technologie chmury i automatyzacji infrastruktury powinny mu pomagać w dostarczaniu bardziej wartościowych produktów w krótszym czasie i w sposób bardziej niezawodny. Ale w praktyce powodują wzrost wielkości, złożoności i różnorodności zarządzanych obiektów.

Technologie te stają się szczególnie istotne wraz z cyfryzacją organizacji. „Cyfryzacja" to termin, za pomocą którego ludzie biznesu tłumaczą, że systemy oprogramowania są niezbędne do tego, co robią ich organizacje[1]. Przejście na technologię cyfrową zwiększa presję, aby robić więcej i szybciej. Trzeba dodawać i obsługiwać więcej usług. Więcej działań biznesowych. Więcej pracowników. Więcej klientów, dostawców i innych interesariuszy.

Narzędzia chmury i automatyzacji są pomocne, ponieważ znacznie ułatwiają dodawanie i zmienianie infrastruktury. Jednak wiele zespołów ma trudności, aby znaleźć dość czasu na nadążenie za już posiadaną infrastrukturą. Ułatwienie tworzenia jeszcze większej liczby rzeczy do zarządzania wcale nie poprawia sytuacji. Jak powiedział mi jeden z klientów: „Użycie chmury zburzyło ściany, za którymi krył się pożar opon"[2].

Wiele osób reaguje na groźbę chaosu zaostrzając procesy zarządzania zmianami. Mają one nadzieję, że zapobiegną chaosowi ograniczając i kontrolując zmiany. W rezultacie zakuwają chmurę w łańcuchy.

Są z tym dwa problemy. Po pierwsze, traci się korzyści płynące z używania technologii chmury; a po drugie, użytkownicy *chcą* czerpać korzyści z tej technologii. Dlatego użytkownicy unikają osób usiłujących ograniczyć chaos. W najgorszych przypadkach ludzie całkowicie ignorują zarządzanie ryzykiem, uznając, że nie ma to znaczenia w nowym,

1 Jest to przeciwieństwo tego, co wiele z tych osób mówiło kilka lat temu o oprogramowaniu, że „nie jest częścią naszej podstawowej działalności". Po zastosowaniu się do tej rady i outsourcingu IT, organizacje zdały sobie sprawę, że zostały wyprzedzone przez inne, zarządzane przez ludzi postrzegających lepsze oprogramowanie jako sposób na konkurowanie, a nie koszty do obcięcia.

2 Jak podaje Wikipedia (*https://oreil.ly/IkDu_*), pożar opon (ang. *tire fire*) miewa dwie formy: „Zdarzenie prowadzące do niemal natychmiastowej utraty kontroli albo powolna piroliza, która może trwać ponad dekadę".

wspaniałym świecie chmury. Efektem tego jest IT rodem z Dzikiego zachodu, co stwarza kolejne problemy[3].

Założeniem tej książki jest to, że można wykorzystać technologię chmury i automatyzacji do łatwego, bezpiecznego, szybkiego i odpowiedzialnego wprowadzania zmian. Te korzyści nie wyskakują z pudełka z narzędziami do automatyzacji ani z platform chmurowych. Zależą od sposobu korzystania z technologii.

DevOps a infrastruktura jako kod

DevOps to ruch mający na celu zredukowanie barier i tarcia między silosami organizacyjnymi – działem rozwoju, operacji i innymi interesariuszami zaangażowanymi w planowanie, tworzenie i uruchamianie oprogramowania. Chociaż technologia jest najbardziej widocznym i pod pewnymi względami najprostszym obliczem DevOps, kultura tego ruchu, ludzie i procesy mają największy wpływ na jego rozwój i skuteczność. Technologia i praktyki inżynierskie, podobnie jak infrastruktura jako kod, powinny być wykorzystywane do wspierania wysiłków na rzecz wypełniania luk i poprawy współpracy.

W tym rozdziale wyjaśniam, że nowoczesna, dynamiczna infrastruktura wymaga nastawienia na „epokę chmury". Takie nastawienie różni się zasadniczo od tradycyjnego podejścia „epoki żelaza", stosowanego w statycznych systemach z czasów poprzedzających chmurę. Definiuję trzy podstawowe praktyki implementowania infrastruktury jako kodu: definiowanie wszystkiego jako kodu, nieustanne testowanie wszystkiego i dostarczanie na bieżąco oraz tworzenie systemu z małych, luźno sprzężonych elementów.

W tym rozdziale opisuję również powody stojące za podejściem „epoki chmury" do infrastruktury. Takie podejście odrzuca fałszywą dychotomię kompromisu między szybkością i jakością. Zamiast tego używa szybkości jako sposobu na poprawę jakości, a jakości jako sposobu na umożliwienie szybkiej dostawy.

Od epoki żelaza do epoki chmury

Technologie epoki chmury umożliwiają szybsze od tradycyjnych, z epoki żelaza, udostępnianie i zmienianie infrastruktury (tabela 1-1).

Jednak technologie te niekoniecznie ułatwiają zarządzanie systemami i ich rozwijanie. Przeniesienie systemu z długiem technicznym[4] do nieograniczonej infrastruktury chmury przyśpiesza chaos.

3 Przez „IT z Dzikiego zachodu" rozumiem osoby budujące systemy IT bez żadnej konkretnej metody ani brania pod uwagę przyszłych konsekwencji. Często ludzie, którzy nigdy nie obsługiwali systemów produkcyjnych, wybierają najkrótszą drogę realizacji bez uwzględniania kwestii bezpieczeństwa, konserwacji, wydajności i innych problemów związanych z obsługą.

4 *https://oreil.ly/3AqHB*

Tabela 1-1 *Zmiany technologii w Epoce chmury*

Epoka żelaza	Epoka chmury
Sprzęt fizyczny	Zasoby wirtualne
Udostępnianie zajmuje tygodnie	Udostępnianie zajmuje minuty
Procesy ręczne	Procesy zautomatyzowane

Być może przydałby się sprawdzony, tradycyjny model nadzoru, aby kontrolować szybkość i chaos uwalniany przez nowsze technologie. Dokładny, wstępny projekt, rygorystyczny przegląd zmian i ściśle rozdzielone obowiązki narzucą porządek! Niestety, modele te są optymalne dla epoki żelaza, w której zmiany są powolne i kosztowne. Przysparzają z góry dodatkową pracę, licząc na skrócenie czasu potrzebnego na późniejsze wprowadzanie zmian. Prawdopodobnie ma to sens, ponieważ późniejsze wprowadzanie zmian jest powolne i kosztowne. Ale chmura sprawia, że zmiany są tanie i szybkie. Należy wykorzystać tę szybkość, aby stale uczyć się i ulepszać swój system. Sposoby pracy w epoce żelaza nakładają ogromny podatek od nauki i doskonalenia.

Zamiast używać powolnych procesów epoki żelaza z szybko zmieniającą się technologią epoki chmury, należy przestawić się na nowy sposób myślenia i wykorzystać szybszą technologię do ograniczenia ryzyka i poprawy jakości. Aby to osiągnąć, konieczna jest fundamentalna zmiana podejścia i nowy sposób myślenia o zmianach i ryzyku (tabela 1-2).

Tabela 1-2 *Sposoby pracy w epoce chmury*

Epoka żelaza	Epoka chmury
Koszt zmian jest wysoki	Koszt zmian jest niski
Zmiany oznaczają porażkę (zmiany muszą być „zarządzane,” „kontrolowane”)	Zmiany oznaczają naukę i ulepszanie
Ograniczenie możliwości niepowodzenia	Maksymalizacja szybkości ulepszania
Dostarczanie dużych porcji, testowanie na końcu	Dostarczanie małych zmian, ciągłe testowanie
Długie cykle wydań	Krótkie cykle wydań
Architektury monolityczne (mniej liczne, większe części)	Architektury mikrousług (bardziej liczne, mniejsze części)
Konfiguracja fizyczna lub oparta na GUI	Konfiguracja jako kod

Infrastruktura jako kod to podejście epoki chmury do zarządzania systemami, które podlegają ciągłym zmianom w celu zapewnienia wysokiej niezawodności i jakości.

Infrastruktura jako kod

Infrastruktura jako kod to podejście do automatyzacji infrastruktury oparte na praktykach zaczerpniętych z programowania. Kładzie ono nacisk na spójne, powtarzalne procedury udostępniania i zmiany systemów i ich konfiguracji. Najpierw wprowadza się

zmiany w kodzie, a potem wykorzystuje automatyzację do testowania i wprowadzenia tych zmian w systemach.

W całej tej książce wyjaśniam, jak wykorzystywać praktyki programowania zwinnego, takie jak TTD (Test Driven Development), CI (Continuous Integration) i CD (Continuous Delivery), aby zmiany infrastruktury były szybkie i bezpieczne. Opisuję również, jak nowoczesny projekt oprogramowania może zapewnić odporną, dobrze utrzymaną infrastrukturę. Takie praktyki i podejścia do projektowania wzajemnie się wzmacniają. Dobrze zaprojektowana infrastruktura jest łatwiejsza do testowania i dostarczania. Zautomatyzowane testowanie i dostarczanie przyczyniają się do prostszych i bardziej czytelnych projektów.

Korzyści z infrastruktury jako kodu

Podsumowując, organizacje wdrażające infrastrukturę jako kod do zarządzania dynamiczną infrastrukturą mają nadzieję na osiągnięcie korzyści, takich jak:

- Wykorzystywanie infrastruktury IT jako czynnika umożliwiającego szybkie dostarczanie
- Zmniejszenie wysiłku i ryzyka wprowadzania zmian w infrastrukturze
- Umożliwienie użytkownikom infrastruktury uzyskania zasobów, gdy będą im potrzebne
- Zapewnienie wspólnych narzędzi dla działu programowania, operacyjnego i innych interesariuszy
- Tworzenie systemów niezawodnych, bezpiecznych i opłacalnych
- Uwidocznienie nadzoru, bezpieczeństwa i kontroli zgodności
- Poprawa szybkości rozwiązywania problemów i usuwania awarii

Używanie infrastruktury jako kodu do optymalizacji pod kątem zmian

Biorąc pod uwagę, że zmiany stanowią największe ryzyko dla systemu produkcyjnego, a ciągłe zmiany są nieuniknione i wprowadzanie ich jest jedynym sposobem na ulepszanie systemu, sensowne jest zoptymalizowanie zdolności do wprowadzania zmian zarówno szybko, jak i niezawodnie. Potwierdzają to badania opisane w raporcie Accelerate State of DevOps. Częste i niezawodne wprowadzanie zmian jest skorelowane z sukcesem organizacyjnym[5].

Kiedy zalecam jakiemuś zespołowi zaimplementowanie automatyzacji w celu optymalizacji pod katem zmian, spotykam się z kilkoma zarzutami. Uważam, że wynikają one z niezrozumienia, jak można i należy stosować automatyzację.

5 Raporty z badań *Accelerate* są dostępne w corocznym raporcie *State of DevOps* (https://oreil.ly/0Q3FE) oraz w książce *Accelerate* autorstwa Dr. Nicole Forsgren, Jez Humble i Gene Kim (IT Revolution Press).

Zarzut: nie dokonujemy zmian tak często, aby była uzasadniona ich automatyzacja

Chcemy myśleć, że jak zbudujemy system, to na tym koniec. Tak patrząc zakładamy, że nie będziemy wprowadzać wielu zmian, więc ich automatyzacja jest stratą czasu.

W rzeczywistości bardzo niewiele systemów przestaje się zmieniać, zanim zostaną wycofane. Niektórzy ludzie uważają, że aktualny poziom zmian jest tymczasowy. Inni z kolei opracowują skomplikowane procesy zgłaszania potrzeby zmian, aby zniechęcić do wysuwania takich żądań. Ci ludzie przeczą faktom. Większość zespołów, które aktywnie wspierają używane systemy, obsługuje ciągły strumień zmian.

Rozważmy typowe przykłady zmian infrastruktury:

- Ważna funkcja nowej aplikacji wymaga dodania nowej bazy danych.
- Nowa funkcja aplikacji wymaga aktualizacji serwera aplikacji.
- Poziom wykorzystania rośnie szybciej niż oczekiwano. Potrzebne są dodatkowe serwery, nowe klastry oraz bardziej rozbudowana sieć i większa pamięć.
- Profilowanie wydajności pokazuje, że obecna architektura wdrażania aplikacji ogranicza wydajność. Trzeba ponownie wdrożyć aplikacje na różnych serwerach aplikacji. To wymaga zmian w klastrach i w architekturze sieci.
- Niedawno ogłoszono lukę w zabezpieczeniach pakietów systemowych naszego systemu operacyjnego. Trzeba zaktualizować dziesiątki serwerów produkcyjnych.
- Trzeba zaktualizować serwery z przestarzałą wersją systemu operacyjnego i pakietami krytycznymi.
- Serwery WWW ulegają sporadycznym awariom. Aby zdiagnozować problem, trzeba wykonać szereg zmian konfiguracyjnych. Następnie trzeba zaktualizować moduł, aby usunąć problem.
- Odkryliśmy zmianę konfiguracji, która poprawi wydajność naszej bazy danych.

Fundamentalna prawda o epoce chmury brzmi: *stabilność jest wynikiem wprowadzania zmian*. Systemy bez poprawek nie są stabilne; są podatne na błędy. Jeśli nie możemy rozwiązywać problemów zaraz po ich wykryciu, to nasz system nie jest stabilny. Jeśli nie możemy przywrócić działania zaraz po awarii, to nasz system nie jest stabilny. Jeśli wprowadzanie zmian powoduje dłuższe przestoje, to nasz system nie jest stabilny. Jeśli zmiany mają często błędy, to nasz system nie jest stabilny.

Zarzut: najpierw trzeba utworzyć, a potem automatyzować

Rozpoczęcie pracy z infrastrukturą jako kodem to stroma krzywa. Skonfigurowanie narzędzi, usług i metod działania w celu zautomatyzowania dostarczania infrastruktury oznacza mnóstwo pracy, zwłaszcza jeśli wdrażamy również nową platformę infrastruktury. Wartość tej pracy jest trudna do wykazania przed rozpoczęciem tworzenia i wdrażania usług za pomocą nowej infrastruktury. A nawet wtedy wartość może nie być widoczna dla osób, które nie pracują bezpośrednio z tą infrastrukturą.

Interesariusze często wywierają presję na zespoły zajmujące się infrastrukturą, aby szybko i ręcznie tworzyły nowe systemy hostowane w chmurze, a ich automatyzację odkładały na później.

Istnieją trzy powody, dla których późniejsza automatyzacja jest złym pomysłem:

- Automatyzacja powinna umożliwiać szybsze dostarczanie, nawet w przypadku nowych rzeczy. Wdrożenie automatyzacji po wykonaniu większości prac powoduje utratę wielu korzyści.
- Automatyzacja ułatwia pisanie zautomatyzowanych testów tego, co robimy. Ułatwia też szybką naprawę i przebudowę w przypadku wykrycia problemów. Wykonanie tego w ramach procesu tworzenia pomaga uzyskać lepszą infrastrukturę.
- Automatyzacja istniejącego systemu jest bardzo trudna. Automatyzacja to część projektowania i implementacji systemu. Aby dodać automatyzację do systemu zbudowanego bez niej, trzeba znacząco zmienić projekt i implementację tego systemu. Dotyczy to również zautomatyzowanego testowania i wdrażania.

Infrastruktura chmury utworzona bez automatyzacji staje się kupą złomu szybciej niż się spodziewamy. Koszt ręcznej konserwacji i naprawy takiego systemu potrafi szybko rosnąć. A jeśli usługa, którą system zapewnia, cieszy się powodzeniem, interesariusze będą naciskać na rozszerzanie i dodawanie funkcji, zamiast pozwolić na przerwę i jego przebudowę.

To samo dotyczy tworzenia systemu w ramach eksperymentu. Gdy mamy już gotowy dowód słuszności naszej koncepcji, pojawia się presja, aby przejść do następnej rzeczy, zamiast cofnąć się i zbudować go dobrze. Tak naprawdę automatyzacja powinna być częścią eksperymentu. Jeśli zamierzamy używać automatyzacji do zarządzania infrastrukturą, musimy rozumieć, jak będzie działać, więc powinna być częścią dowodu słuszności koncepcji.

Rozwiązaniem jest stopniowe tworzenie systemu, z automatyzacją na bieżąco. Należy zapewnić stały strumień wartości, jednocześnie budując możliwość robienia tego w sposób ciągły.

Zarzut: musimy wybierać między szybkością i jakością

Myślenie, że można poruszać się szybko tylko kosztem jakości, a jakość można zapewnić tylko poruszając się powoli, jest całkiem naturalne. Można postrzegać to jako kontinuum przedstawione na rysunku 1-1.

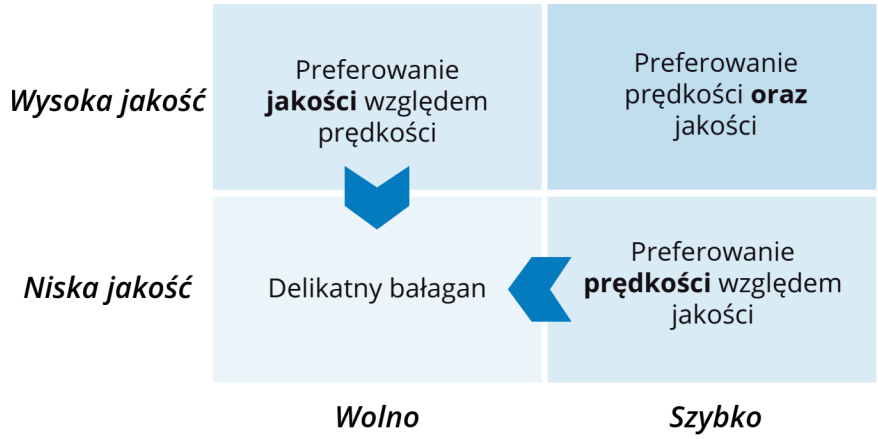
Rysunek 1-1 *Pomysł, że szybkość i jakość są na dwóch przeciwległych końcach widma, jest fałszywą dychotomią*

Jednak badanie *Accelerate*, o którym wspomniałem wcześniej („Używanie infrastruktury jako kodu do optymalizacji pod kątem zmian" na stronie 6) dowodzi czegoś przeciwnego:

> Wyniki te pokazują, że nie ma kompromisu między poprawą wydajności a osiągnięciem wyższego poziomu stabilności i jakości. W rzeczywistości dobrzy wykonawcy radzą sobie lepiej pod każdym względem. To jest dokładnie to, co przewidują ruchy Agile i Lean, ale wiele dogmatów w naszej branży nadal opiera się na fałszywym założeniu, że poruszanie się szybciej odbywa się kosztem innych aspektów wydajności, a nie umożliwia je i wspiera.
>
> *dr Nicole Forsgren, Accelerate*

Krótko mówiąc, organizacje nie wybierają między optymalizacją pod kątem zmian a stabilnością. W efekcie są albo dobre w jednym i drugim albo w obu złe.

Wolę postrzegać jakość i szybkość jako kwadrant niż kontinuum[6], jak to jest pokazane na rysunku 1-2.



Rysunek 1-2 *Kwadrant szybkości i jakości*

Ten model kwadrantu pokazuje, dlaczego próba wyboru między szybkością i jakością prowadzi do przeciętności pod obu względami:

Prawa dolna ćwiartka: ważniejsza jest szybkość niż jakość
: To jest filozofia „działaj szybko i taranuj". Zespoły stosujące optymalizację pod kątem szybkości kosztem jakości tworzą niechlujne i kruche systemy. Wpadają do lewego dolnego kwadrantu, ponieważ ich tandetne systemy spowalniają ich pracę. Wiele startupów, które od jakiegoś czasu działają w ten sposób, narzeka na utratę „atrakcyjności". Proste zmiany, które w danych czasach wymagałyby chwili, teraz zajmują dnie i tygodnie, ponieważ system jest zagmatwanym kłębkiem.

6 Tak, pracuję w firmie konsultingowej, dlaczego pytasz?

Lewa górna ćwiartka: ważniejsza jest jakość niż szybkość
: Podejście znane również jako „robimy rzeczy poważne i doniosłe, musimy więc robić je *właściwie*". Ale presja terminu wykonania zmusza do „obejść". Duże procesy stawiają bariery przed ulepszaniem, więc dług techniczny rośnie wraz z listami „znanych problemów". Takie zespoły wpadają do lewego dolnego kwadrantu. Kończą z systemami niskiej jakości, *ponieważ* zbyt trudno je ulepszać. Generują jeszcze więcej procesów w odpowiedzi na awarie. Procesy te jeszcze bardziej utrudniają wprowadzanie ulepszeń i zwiększają kruchość i ryzyko. Prowadzi to do zwiększenia liczby niepowodzeń i procesów. Wiele osób pracujących w organizacjach działających w ten sposób uznaje to za normalne[7], zwłaszcza ci, którzy pracują w branżach uwzględniających ryzyko[8].

Prawy górny kwadrant jest celem nowoczesnego podejścia, takiego jak Lean, Agile czy DevOps. Możliwość szybkiego poruszania się i utrzymania wysokiego poziomu jakości może wydawać się fantazją. Jednak badanie Accelerate udowadnia, że wiele zespołów potrafi to osiągnąć. W tym kwadrancie można znaleźć „najlepszych wykonawców".

Cztery kluczowe wskaźniki

Zespół badawczy DORA z *Accelerate* wymienia cztery kluczowe wskaźniki dostarczania oprogramowania i wydajności operacyjnej[9]. Badanie obejmowało różne miary i okazało się, że te cztery mają najsilniejszy związek z tym, jak dobrze organizacja realizuje swoje cele:

Czas realizacji dostawy
: Czas potrzebny na implementację, przetestowanie i dostarczenie zmian do systemu produkcyjnego

Częstotliwość wdrażania
: Jak często są wdrażane zmiany w systemach produkcyjnych

Procent nieudanych zmian
: Jaki procent zmian powoduje pogorszenie jakości usługi lub konieczność natychmiastowej korekty, takiej jak wycofanie lub poprawka awaryjna

Średni czas naprawy (Mean Time to Repair – MTTR)
: Ile czasu zajmuje przywrócenie usługi w przypadku nieplanowanego przestoju lub jej utraty

---

7 Jest to przykład „normalizacji dewiacji", ponieważ ludzie przyzwyczajają się do sposobów pracy zwiększających ryzyko. Diane Vaughan zdefiniowała ten termin w *The Challenger Launch Decision* (University Of Chicago Press).

8 To ironiczne (i przerażające), że tak wielu ludzi w branżach takich jak finanse, administracja i opieka zdrowotna uważają delikatne systemy IT – oraz procesy, które utrudniają ich ulepszanie – za normalne, a nawet pożądane.

9 DORA, obecnie część firmy Google, jest zespołem, który opracował raport *Accelerate State of DevOps* (*https://oreil.ly/ysk9n*).

Organizacje, które osiągają dobre wyniki w realizacji swoich celów – bez względu na to, czy chodzi o przychód, cenę akcji czy inne kryteria – równie dobrze wypadają pod względem tych czterech wskaźników i na odwrót. Pomysły zawarte w tej książce mają na celu pomóc zespołom i organizacjom osiągnąć dobre wartości tych wskaźników. Ułatwią to trzy podstawowe praktyki dotyczące infrastruktury jako kodu.

Trzy podstawowe praktyki dotyczące infrastruktury jako kodu

Koncepcja epoki chmury wykorzystuje dynamiczny charakter nowoczesnej infrastruktury i platform aplikacji do częstego i niezawodnego wprowadzania zmian. Infrastruktura jako kod to podejście do tworzenia infrastruktury, które obejmuje ciągłe zmiany jako środek do zapewnienia wysokiej niezawodności i jakości. Jak więc można to osiągnąć?

Są trzy podstawowe praktyki wdrażania infrastruktury jako kodu:

- Definiowanie wszystkiego jako kodu
- Stałe testowanie i dostarczanie wszystkiego na bieżąco
- Tworzenie małych, prostych elementów, które można zmieniać niezależnie

Podsumuję teraz każdą z nich, aby przygotować kontekst do dalszej dyskusji. Później poświęcę rozdział zasadom wdrażania każdej z tych praktyk.

Podstawowa praktyka: definiowanie wszystkiego jako kodu

Definiowanie wszystkiego „jako kodu” jest podstawową praktyką umożliwiającą szybkie i niezawodne wprowadzanie zmian. Jest kilka powodów, dla których to pomaga:

Możliwość wielokrotnego wykorzystania
: Jeśli jakaś rzecz jest zdefiniowana jako kod, to można utworzyć wiele jej instancji. Można szybko naprawić i ponownie utworzyć taką rzecz, a inni mogą tworzyć jej identyczne instancje.

Spójność
: Rzeczy zbudowane z kodu są za każdym razem budowane tak samo. To sprawia, że zachowanie systemu jest przewidywalne, testowanie bardziej niezawodne, a do tego możliwe jest ciągłe testowanie i dostarczanie.

Przejrzystość
: Patrząc na kod każdy może przekonać się, jak jest zbudowana dana rzecz. Ludzie mogą przeglądać kod i sugerować ulepszenia. Mogą dowiedzieć się, jak wykorzystać tę rzecz w innym kodzie, jak jej użyć do rozwiązywania problemów, a ponadto wykonać inspekcję pod kątem zgodności.

O koncepcjach i zasadach implementacji w przypadku definiowania rzeczy jako kodu opowiem szerzej w rozdziale 4.

Podstawowa praktyka: stałe testowanie i dostarczanie wszystkiego na bieżąco

Efektywne zespoły infrastruktury podchodzą rygorystycznie do testowania. Używają automatyzacji do wdrażania i testowania każdego składnika swojego systemu oraz integrują efekty pracy, którą wszyscy wykonują. Testują na bieżąco, zamiast czekać na koniec.

Chodzi o to, aby budować jakość, a nie próbować *testować jakość*.

Jednym z elementów, o którym ludzie często zapominają, jest integracja i testowanie wszystkich prac na bieżąco. W wielu zespołach ludzie pracują nad kodem w oddzielnych działach i integrują go dopiero po zakończeniu. Z badań Accelerate wynika jednak, że zespoły osiągają lepsze wyniki, gdy każdy przynajmniej co dzień integruje swoją pracę. CI stosuje scalanie i testowanie kodu wszystkich osób w trakcie programowania. CD idzie dalej, utrzymując scalony kod zawsze gotowy do produkcji.

Ciągłe testowanie i dostarczanie kodu infrastruktury omówię bardziej szczegółowo w rozdziale 8.

Podstawowa praktyka: tworzenie małych, prostych elementów, które można zmieniać niezależnie

Zespoły borykają się z problemami, gdy ich systemy stają się duże i silnie sprzężone. Im większy system, tym trudniej go zmienić i tym łatwiej go popsuć.

Gdy patrzymy na bazę kodu wydajnego zespołu, widzimy różnicę. System składa się z małych, prostych elementów. Każdy element jest zrozumiały i ma wyraźnie określone interfejsy. Zespół może łatwo zmienić każdy składnik niezależnie od innych oraz wdrożyć go i oddzielnie przetestować.

Zasady implementacji tej podstawowej praktyki omówię bardziej szczegółowo w rozdziale 15.

Podsumowanie

Aby czerpać korzyści z chmury i automatyzacji infrastruktury, potrzebne jest podejście z epoki chmury. Oznacza to wykorzystywanie szybkości do poprawy jakości i budowanie jakości w celu zwiększenia szybkości. Automatyzacja infrastruktury wymaga pracy, zwłaszcza gdy dopiero uczymy się to robić. Ale takie działanie pomaga wprowadzić zmiany, w tym przede wszystkim zbudować system.

Opisałem części typowego systemu infrastruktury, ponieważ stanowią one podstawę rozdziałów, w których wyjaśnię, jak wdrażać infrastrukturę jako kod.

Na koniec zdefiniowałem trzy podstawowe praktyki dotyczące infrastruktury jako kodu: definiowanie wszystkiego jako kodu, ciągłe testowanie i dostarczanie oraz tworzenie małych elementów.

Rozdział 2

Zasady infrastruktury w epoce chmury

Zasoby komputerowe w epoce żelaza branży IT były silnie sprzężone ze sprzętem fizycznym. Wkładaliśmy procesor, pamięć i dyski twarde do obudowy, umieszczaliśmy ją w szafie rackowej i podłączaliśmy do przełączników i routerów. Instalowaliśmy i konfigurowaliśmy system operacyjny oraz oprogramowanie użytkowe. Umieliśmy wskazać, w którym miejscu centrum danych znajduje się serwer aplikacji: na którym piętrze, w którym rzędzie, w której szafie, na jakiej wysokości.

Epoka chmury oddzieliła zasoby obliczeniowe od sprzętu fizycznego, na którym działają. Sprzęt oczywiście nadal istnieje, ale serwery, dyski twarde i routery są rozsiane po różnych miejscach. Nie ma już rzeczy fizycznych- zostały one przekształcone w wirtualne konstrukcje, które tworzymy, duplikujemy, zmieniamy lub usuwamy wedle uznania.

Ta transformacja zmusiła nas do zmiany sposobu myślenia, projektowania i wykorzystywania zasobów obliczeniowych. Nie możemy zakładać, że fizyczne atrybuty naszego serwera aplikacji będą niezmienne. Musimy być w stanie szybko dodawać i usuwać instancje naszych systemów oraz łatwo utrzymywać ich spójność i jakość, nawet w przypadku ich gwałtownego rozrostu.

Istnieje kilka zasad projektowania i wdrażania infrastruktury w platformach chmurowych. Zasady te wyjaśniają powody stosowania trzech podstawowych praktyk (definiowanie wszystkiego jako kodu, ciągłe testowanie i dostarczanie, tworzenie małych elementów). Wymieniam również kilka typowych pułapek, które czyhają na zespoły w przypadku dynamicznej infrastruktury.

Te zasady i pułapki leżą u podstaw bardziej szczegółowych porad dotyczących praktyk implementowania infrastruktury jako kodu w całej tej książce.

Zasada: zakładaj, że systemy są zawodne

W epoce żelaza zakładaliśmy, że nasze systemy działają na niezawodnym sprzęcie. W epoce chmury musimy zakładać, że nasz system działa na zawodnym sprzęcie[1].

1 Nauczyłem się tego z artykułu Sama Johnsona „Simplifying Cloud: Reliability” (*https://oreil.ly/S3VRT*).

Infrastruktura chmury obejmuje setki tysięcy urządzeń, jeśli nie więcej. W tej skali awarie zdarzają się nawet w przypadku używania niezawodnego sprzętu – a większość dostawców chmury wykorzystuje tani, mniej niezawodny sprzęt, który po wykryciu awarii jest wymieniany na nowy.

Trzeba przełączać fragmenty systemów w tryb offline nie tylko z powodu nieplanowanych awarii. Trzeba łatać i aktualizować systemy. Trzeba zmieniać wielkość, rozkładać obciążenie i rozwiązywać problemy.

W przypadku statycznej infrastruktury robienie tych rzeczy oznacza przełączanie systemów w tryb offline. Ale dla wielu współczesnych organizacji przejście systemów w tryb offline oznacza przejście biznesu w tryb offline.

Nie można więc traktować infrastruktury, w której działa system, jako stabilnego fundamentu. Zamiast tego trzeba tak projektować, aby zapewnić nieprzerwane świadczenie usług w przypadku zmiany podstawowych zasobów[2].

Zasada: rób tak, aby wszystko było odtwarzalne

Jednym ze sposobów zapewnienia odtwarzalności systemu jest zadbanie, aby można było zawsze odbudować jego fragmenty, bez wysiłku i niezawodnie.

Bez wysiłku oznacza, że nie trzeba podejmować żadnych decyzji na temat tego, jak należy odbudowywać. Trzeba tylko zdefiniować ustawienia konfiguracji, wersje oprogramowania i zależności jako kod. Odbudowa jest wtedy prostą decyzją „tak/nie".

Odtwarzalność nie tylko ułatwia odzyskanie uszkodzonego systemu, ale także pomaga:

- Zapewnić zgodność środowisk testowych z produkcyjnymi
- Replikować systemy w różnych regionach, aby zapewnić dostępność
- Dodawać instancje na żądanie, aby poradzić sobie z dużym obciążeniem
- Replikować systemy, aby zapewnić każdemu klientowi dedykowaną instancję

Oczywiście system generuje dane, zawartość i dzienniki, których nie można zdefiniować z wyprzedzeniem. Trzeba je zidentyfikować i znaleźć sposoby zachowywania w ramach swojej strategii replikacji. Może to być tak proste, jak kopiowanie lub strumieniowanie danych do kopii zapasowej, a następnie przywracanie ich podczas odbudowy. Opiszę opcje, jak to robić, w podrozdziale „Ciągłość danych w zmieniającym się systemie" na stronie 373.

Możliwość budowania i przebudowywania bez wysiłku dowolnej części infrastruktury ma bardzo duże znaczenie. Eliminuje ryzyko i lęk przed wprowadzaniem zmian i pozwala bez obaw radzić sobie z awariami. Umożliwia szybkie udostępnianie nowych środowisk i usług.

2 Zasada zakładania, że systemy są zawodne, jest motorem inżynierii chaosu (*https://oreil.ly/7fvio*), w której w sposób kontrolowany wywołuje się awarie, aby przetestować i podnieść niezawodność usług. Mówię o tym więcej w punkcie „Inżynieria chaosu" na stronie 371.

Pułapka: systemy śnieżynki

Śnieżynka to instancja systemu lub części systemu, którą trudno odbudować. Może to być również środowisko, które powinno być podobne do innych środowisk, takie jak środowisko tymczasowe, ale różni się w sposób, którego zespół w pełni nie rozumie.

Ludzie nie chcą budować systemów śnieżynek. Są one zjawiskiem naturalnym. Tworząc coś po raz pierwszy za pomocą nowego narzędzia uczymy się przy okazji, z czym wiąże się popełnianie błędów. Ale jeśli ludzie zaczną już używać zbudowanej przez nas rzeczy, możemy nie mieć czasu, aby się cofnąć i przebudować ją lub ulepszyć na podstawie tego, czego się nauczyliśmy. Ulepszanie już zbudowanych rzeczy jest szczególnie trudne, gdy nie ma mechanizmów i praktyk sprawiających, że taka zmiana jest łatwa i bezpieczna.

Innym powodem występowania śnieżynek jest to, że ludzie wprowadzają zmiany w jednej instancji systemu, a nie robią tego w pozostałych. Mogą być pod presją, aby usunąć problem, który występuje tylko w jednym systemie, albo mogą rozpocząć dużą aktualizację w środowisku testowym, ale nie mieć czasu na wdrożenie jej w innych.

System jest śnieżynką, jeśli nie mamy pewności, że można go bezpiecznie zmienić lub ulepszyć. Co gorsza, jeśli system się zepsuje, trudno go naprawić. Dlatego ludzie unikają wprowadzania zmian w takim systemie, przez co pozostaje on nieaktualny, bez poprawek, a może nawet częściowo popsuty.

Systemy śnieżynki stwarzają ryzyko i marnują czas zarządzających nimi zespołów. Prawie zawsze warto spróbować zastąpić je systemami odtwarzalnymi. Jeśli system śnieżynka nie jest wart ulepszania, to może nie jest w ogóle wart zachowania.

Najlepszym sposobem na zastąpienie systemu śnieżynki jest napisanie kodu, który potrafi replikować system, uruchamiając równolegle nowy system i czekając, aż będzie gotowy. Należy skorzystać ze zautomatyzowanych testów i potoków, aby udowodnić, że nowy system jest poprawny i odtwarzalny oraz że można go łatwo zmienić.

Zasada: twórz rzeczy zastępowalne

Budowa systemu, który radzi sobie z dynamiczną infrastrukturą, to pierwszy poziom. Następny poziom to budowa systemu, który sam w sobie jest dynamiczny. Taki system powinien umożliwiać eleganckie dodawanie, usuwanie, uruchamianie, zatrzymywanie, zmienianie i przenoszenie jego części. W ten sposób zapewniamy mu elastyczność operacyjną, dostępność i skalowalność. Ponadto upraszczamy dokonywanie zmian i zmniejszamy towarzyszące im ryzyko.

Zapewnienie elastyczności elementów systemu to główna idea natywnego oprogramowania chmury. Chmura oddziela zasoby infrastruktury (obliczeniowe, sieciowe i pamięciowe) od sprzętu fizycznego. Natywne oprogramowanie chmury całkowicie rozdziela funkcjonalność aplikacji i infrastrukturę, na której działa[3].

3 Zobacz „Infrastruktura natywna dla chmury i oparta na aplikacjach” na stronie 150

Bydło hodowlane, a nie domowi ulubieńcy

„Traktuj swoje serwery jak bydło hodowlane, a nie ulubieńców domowych" to popularne wyrażenie odnoszące się do zastępowalności[4]. Brakuje mi nadawania zabawnych imion wszystkim tworzonym przeze mnie serwerom. Ale nie marzę o konieczności dopieszczania każdego serwera w naszym środowisku.

Jeśli systemy są dynamiczne, to narzędzia używane do zarządzania nimi muszą sobie z tym radzić. Na przykład monitoring nie powinien podnosić alarmu za każdym razem, gdy przebudowujemy część naszego systemu. Natomiast powinien zgłosić ostrzeżenie, jeśli coś zacznie odbudowywać się w pętli.

Przypadek znikającego serwera plików

Ludzie potrzebują zwykle trochę czasu, żeby przyzwyczaić się do ulotnej infrastruktury. Pracowałem z zespołem, który skonfigurował zautomatyzowaną infrastrukturę za pomocą narzędzi VMware i Chef. W razie potrzeby zespół usuwał i odbudowywał maszyny wirtualne.

Nowy programista w zespole potrzebował serwera do hostingu plików i udostępniania ich członkom zespołu, więc ręcznie zainstalował serwer HTTP na serwerze programistycznym i tam umieścił pliki. Kilka dni później odbudowałem maszynę wirtualną i jego serwer WWW zniknął.

Po chwili zdezorientowania programista zrozumiał, dlaczego tak się stało. Dodał swój serwer WWW do kodu Chefa i zamieścił swoje pliki w sieci SAN. Zespół miał teraz niezawodną usługę udostępniania plików.

Zasada: minimalizuj zróżnicowanie

Wraz z rozwojem systemu coraz trudniej jest go zrozumieć, zmieniać i naprawiać. Wymagana praca rośnie wraz z liczbą elementów, a także z liczbą ich rodzajów. Dlatego dobrym sposobem na utrzymanie możliwości zarządzania systemem jest dbanie o małą liczbę elementów – małą różnorodność. Łatwiej jest zarządzać stu identycznymi serwerami niż pięcioma zupełnie różnymi serwerami.

Zasada odtwarzalności (zobacz „Zasada: rób tak, aby wszystko było odtwarzalne" na stronie 14) uzupełnia tę ideę. Jeśli zdefiniujemy prosty komponent i utworzymy wiele identycznych jego instancji, będzie można go łatwo zrozumieć, zmieniać i naprawiać.

4 Po raz pierwszy usłyszałem to wyrażenie podczas prezentacji Gavina McCance'a „CERN Data Centre Evolution" (*https://oreil.ly/cDt47*). Randy Bias ceni prezentację Billa Bakera „Architectures for Open and Scalable Clouds" (*https://oreil.ly/_SG96*). Obie te prezentacje stanowią doskonałe wprowadzenie do tych zasad.

Aby to działało, należy stosować wszelkie zmiany do wszystkich instancji danego komponentu. W przeciwnym razie powstaje dryf konfiguracji.

Oto kilka rodzajów odmian, które występują w systemach:

- Wiele systemów operacyjnych, środowisk wykonawczych aplikacji, baz danych i innych technologii. Każdy z tych składników wymaga w zespole ludzi z odpowiednimi umiejętnościami i wiedzą.
- Wiele wersji oprogramowania, takiego jak system operacyjny czy baza danych. Nawet jeśli jest używany tylko jeden system operacyjny, to i tak każda wersja może wymagać innej konfiguracji i narzędzi.
- Różne wersje pakietów. Gdy niektóre serwery mają nowszą wersję pakietu, narzędzia lub biblioteki niż inne, pojawia się ryzyko. Polecenia mogą nie działać jednakowo we wszystkich wersjach, a starsze wersje mogą zawierać luki lub błędy.

Organizacje stoją przed dylematem, czy pozwalać każdemu zespołowi na wybór technologii i rozwiązań, które są odpowiednie do jego potrzeb, czy też utrzymywać stopień zróżnicowania w organizacji na możliwym do zarządzania poziomie.

Lekki nadzór

Nowoczesne, cyfrowe organizacje uczą się wartości *Lekkiego nadzorowania* IT, które równoważy autonomię i scentralizowane sterowanie. Jest to kluczowy element modelu EDGE dla zwinnych organizacji. Więcej na ten temat można znaleźć w książce *EDGE: Value-Driven Digital Transformation*[5] Jima Highsmitha, Lindy Luu i Davida Robinsona (Addison-Wesley Professsional) oraz w wystąpieniu Jonny'ego LeRoy'a „The Goldilocks Zone of Lightweight Architectural Governance"[6].

Dryf konfiguracji

Dryf konfiguracji to różnice pojawiające się z czasem w systemach, które kiedyś były identyczne. Widać to na rysunku 2-1. Dryf konfiguracji może być efektem ręcznych zmian. Może również nastąpić, jeśli używamy narzędzi automatyzacji do wprowadzania zmian ad hoc tylko w niektórych instancjach. Dryf konfiguracji utrudnia zachowanie spójnej automatyzacji.

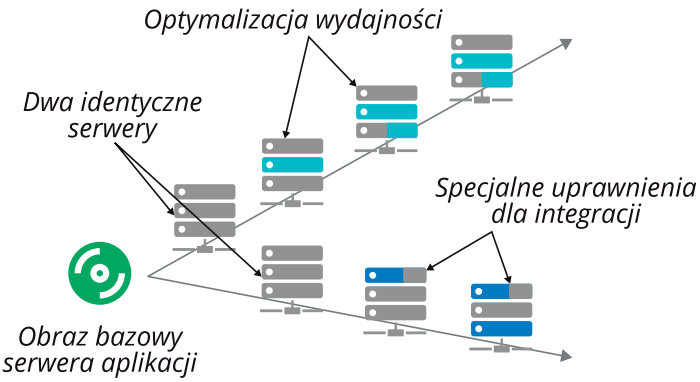
Jako przykład dywergencji konfiguracji w czasie rozważmy przypadek naszego przykładowego zespołu firmy ShopSpinner[7].

ShopSpinner uruchamia osobną instancję swojej aplikacji dla każdego sklepu, przy czym każda instancja jest tak skonfigurowana, aby korzystała z niestandardowego brandingu i zawartości katalogu produktów. Na początku zespół ShopSpinner uruchamiał

5 *https://oreil.ly/Eg3Mu*

6 *https://oreil.ly/d29qg*

7 ShopSpinner to fikcyjna firma, która pozwala innym firmom zakładać i prowadzić sklepy internetowe. Będę używał jej w całej tej książce do ilustrowania koncepcji i praktyk.



Rysunek 2-1 *Dryf konfiguracji ma miejsce w sytuacji, gdy wystąpienia tej samej rzeczy z upływem czasu zaczynają się różnić*

skrypty, aby stworzyć nowy serwer aplikacji dla każdego nowego sklepu. Zespół zarządzał infrastrukturą ręcznie albo pisząc skrypty i dostrajając je za każdym razem, gdy była potrzebna zmiana.

Jeden z klientów, Water Works[8], ma znacznie większy ruch w swojej aplikacji do zarządzania zamówieniami niż pozostali, więc zespół dostroił konfigurację serwera dla Water Works. Zmiany nie zostały wprowadzone u innych klientów, ponieważ zespół był zajęty i nie uznał tego za konieczne.

Później zespół ShopSpinner przystosował narzędzie Servermaker do automatyzacji konfiguracji swojego serwera aplikacji[9]. Najpierw przetestowano serwer dla Palace Pens[10], mniejszego klienta, a następnie udostępniono pozostałym. Niestety, kod nie uwzględniał optymalizacji wydajności dla Water Works, więc ulepszenia przepadły. Serwer Water Works zwolnił znacząco, aż zespół zauważył to i naprawił błąd.

Sposobem na rozwiązanie problemu okazała się parametryzacja kodu Servermakera. Teraz można ustawiać różne poziomy zasobów dla poszczególnych klientów. W ten sposób zespół może nadal stosować jednakowy kod dla wszystkich, optymalizując go jednocześnie dla każdego klienta. W rozdziale 7 są opisane niektóre wzorce i antywzorce parametryzacji kodu infrastruktury dla różnych instancji.

Zasada: pilnuj, abyś mógł powtórzyć każdy proces

Jeśli stosujemy zasadę odtwarzalności, powinniśmy być w stanie powtórzyć wszystko, co robimy z naszą infrastrukturą. Łatwiej jest powtarzać czynności za pomocą skryptów i narzędzi do zarządzania konfiguracją, niż robić to ręcznie. Ale automatyzacja może wymagać dużo pracy, zwłaszcza jeśli nie jesteśmy do tego przyzwyczajeni.

8 Water Works wysyła co miesiąc butelki z wodą innego wytwórcy.
9 Servermaker to fikcyjne narzędzie do konfigurowania serwerów, podobne do Ansible, Chef i Puppet.
10 Palace Pens sprzedaje najlepsze na świecie luksusowe przybory do pisania.

Spirala strachu przed automatyzacją

Spirala strachu przed automatyzacją opisuje, jak wiele zespołów wpada w dryf konfiguracji i dług techniczny.

Na sesji poświęconej automatyzacji konfiguracji podczas konferencji DevOpsDays (*https://oreil.ly/x8G0C)* zapytałem grupę, ile osób korzysta z narzędzi automatyzacji, takich jak Ansible, Chef czy Puppet. Większość podniosła rękę w górę. Zapytałem, ile osób uruchamia te narzędzia bez nadzoru, używając automatycznego harmonogramu. Większość opuściła rękę.

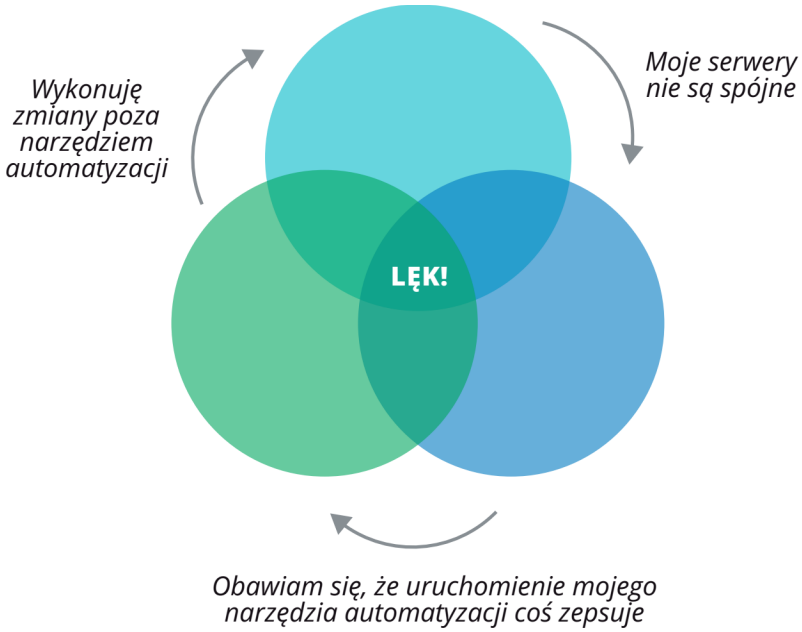
Wiele osób ma ten sam problem, który miałem na początku korzystania z narzędzi do automatyzacji. Używałem automatyzacji wybiórczo – na przykład, aby tworzyć nowe serwery lub dokonać określonej zmiany konfiguracji. Dostrajałem konfigurację za każdym razem, gdy ją uruchamiałem, aby odpowiadała konkretnemu zadaniu, które wykonywałem.

Bałem się zostawiać moje narzędzia automatyzacji bez nadzoru, ponieważ nie miałem pewności, co zrobią.

Brakowało mi zaufania do mojej automatyzacji, ponieważ moje serwery nie były spójne.

Moje serwery nie były spójne, ponieważ nie uruchamiałem automatyzacji dość często ani konsekwentnie.

To właśnie jest spirala lęku przed automatyzacją, pokazana na rysunku 2-2. Zespoły ds. infrastruktury muszą przerwać tę spiralę, aby skutecznie używać automatyzacji.



Rysunek 2-2 *Spirala lęku przed automatyzacją*

Najskuteczniejszym sposobem przerwania spirali jest zmierzenie się z własnymi lękami. Trzeba zacząć od jednego zestawu serwerów. Upewnić się, że można zastosować, a następnie ponownie zastosować kod infrastruktury na tych serwerach. Następnie zaplanować godzinowy proces, który będzie w sposób ciągły stosować kod do tych serwerów. Następnie wybrać inny zestaw serwerów i powtórzyć proces. I kontynuować, aż każdy serwer będzie ciągle aktualizowany.

Dobry monitoring i zautomatyzowane testowanie budują pewność ciągłej synchronizacji kodu. A ta ujawnia dryf konfiguracji, gdy ma miejsce, więc można go szybko naprawić.

Przypuśćmy, na przykład, że muszę jednorazowo podzielić dysk na partycje. Napisanie i przetestowanie skryptu wymaga znacznie więcej pracy niż po prostu zalogowanie się i uruchomienie polecenia `fdisk`. Dlatego robię to ręcznie.

Problem pojawia się później, gdy ktoś z mojego zespołu, Priya, musi podzielić inny dysk. Dochodzi do tego samego wniosku co ja i robi to ręcznie, zamiast napisać skrypt. Podejmuje jednak nieco inną decyzję co do sposobu podziału dysku. Ja zrobiłem na moim serwerze partycję `/var` typu ext3 80 GB, a Priya tworzy u siebie partycję typu xjs 100 GB. Powodujemy dryf konfiguracyjny, który osłabia naszą zdolność do zaufanej automatyzacji.

Efektywne zespoły infrastruktury cechuje silna kultura skryptowa. Jeśli możesz napisać skrypt dla zadania, napisz go[11]. Jeśli napisać skrypt jest trudno, zbadaj problem dokładniej. Może jest jakaś technika lub narzędzie, które mogą pomóc, a może da się uprościć zadanie albo podejść do niego inaczej. Rozbijanie pracy na zadania realizowane za pomocą skryptów sprawia, że całość staje się prostsza, bardziej przejrzysta i bardziej niezawodna.

Podsumowanie

Zasady infrastruktury w epoce chmury odzwierciedlają różnice między tradycyjną, statyczną infrastrukturą, a nowoczesną, dynamiczną infrastrukturą:

- zakładaj, że systemy są zawodne
- rób tak, aby wszystko było odtwarzalne
- twórz rzeczy zastępowalne
- minimalizuj zróżnicowanie
- pilnuj, abyś mógł powtórzyć każdy proces

Zasady te stanowią klucz do wykorzystania natury platform chmurowych. Zamiast wzbraniać się przed możliwością robienia zmian przy minimalnym wysiłku, wykorzystaj tę możliwość do zapewnienia jakości i niezawodności.

11 Mój kolega Florian Sellmayr powiada „jeśli warto to dokumentować, to warto też automatyzować”.

Rozdział 3

Platformy infrastruktury

Istnieje szeroki wachlarz narzędzi związanych na różne sposoby z nowoczesną infrastrukturą chmury. Pytanie, które technologie wybrać i jak je połączyć, może być przytłaczające. W tym rozdziale przedstawiam model myślenia o problemach związanych z platformą na wyższym poziomie, jej możliwościach oraz zasobach infrastruktury, które można gromadzić, aby zapewnić te możliwości.

To nie jest autorytatywny model ani architektura. Czytelnik może mieć własny sposób na opisanie części swojego systemu. Ustalenie właściwego miejsca na diagramie dla każdego używanego narzędzia lub technologii jest mniej ważne niż rozmowa.

Celem tego modelu jest stworzenie kontekstu do dyskusji o pojęciach, praktykach i podejściach przedstawionych w tej książce. Szczególnie ważne jest upewnienie się, że te dyskusje są odpowiednie dla Czytelnika, niezależnie od stosu technologii, narzędzi lub platform, których używa. Dlatego model definiuje sposoby grupowania i terminologię, której w późniejszych rozdziałach używam do opisu, na przykład, udostępniania serwerów na platformie wirtualizacyjnej, takiej jak VMware lub w chmurze IaaS, takiej jak AWS.

Części systemu infrastruktury

Nowoczesna infrastruktura chmury składa się z wielu różnych części wielu różnych typów. Uważam, że wygodnie jest zgrupować te części w trzy warstwy platformy (rysunek 3-1):

Aplikacje
: Aplikacje i usługi zapewniają organizacji i jej użytkownikom różnego rodzaju możliwości. Cała reszta tego modelu istnieje po to, żeby służyć tej warstwie.

Środowiska wykonawcze aplikacji
: Środowiska wykonawcze aplikacji zapewniają usługi i różne możliwości warstwie aplikacji. Przykładami usług i konstrukcji na platformie wykonawczej aplikacji są klastry kontenerów, środowiska bezserwerowe, serwery aplikacji, systemy operacyjne i bazy danych. Warstwa ta nosi też nazwę platforma jako usługa (Platform as a Service – PaaS).