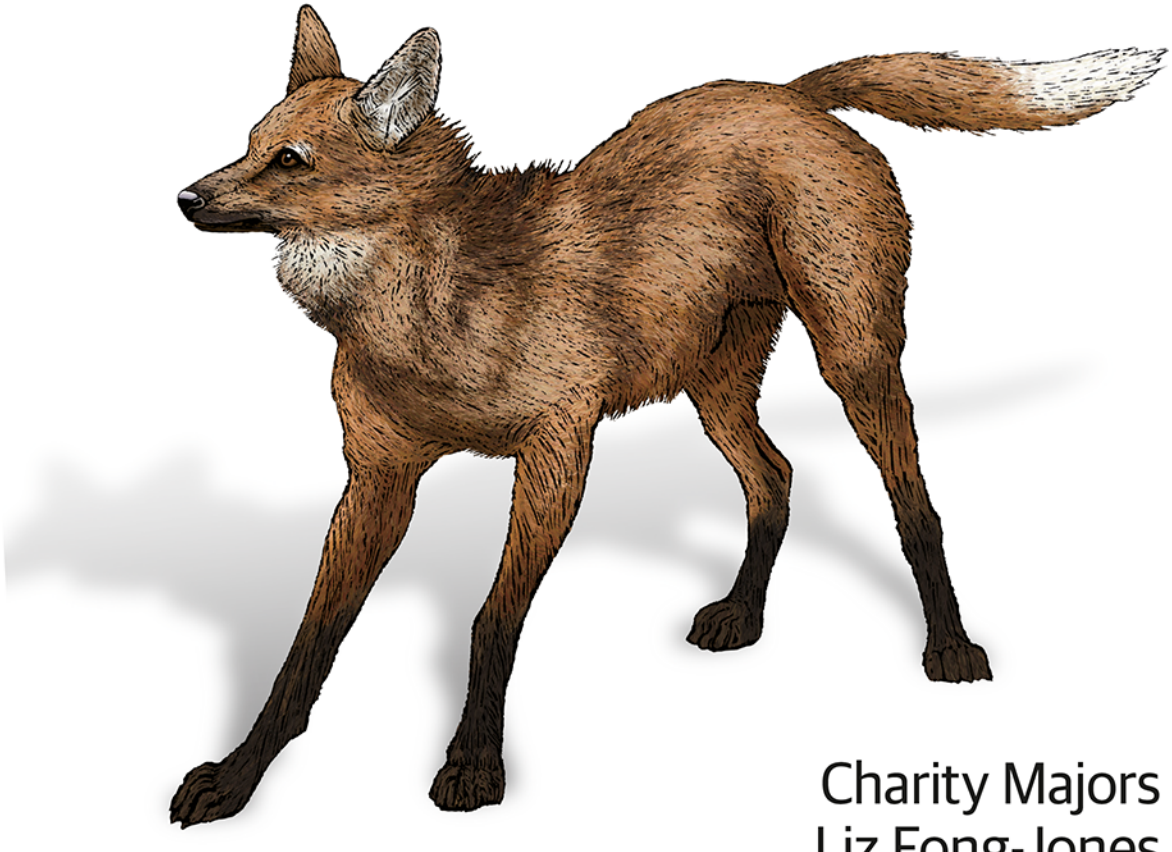
O'REILLY®

Inżynieria obserwowalności

Doskonalenie produkcyjnych systemów oprogramowania



Charity Majors
Liz Fong-Jones
George Miranda

Helion

Tytuł oryginału: Observability Engineering: Achieving Production Excellence

Tłumaczenie: Tomasz Walczak

ISBN: 978-83-289-1198-7

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

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
https://ftp.helion.pl/przyklady/inzobs.zip

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

Printed in Poland.

Spis treści

ROZDZIAŁ 1.

Czym jest obserwowalność?

W branży rozwoju oprogramowania temat obserwowalności cieszy się dużym zainteresowaniem i często pojawia się na listach nowych, gorących tematów. Ale jak to nieuchronnie bywa, gdy nowy, popularny temat cieszy się rosnącym zainteresowaniem, złożone idee stają się narażone na niezrozumienie bez głębszego spojrzenia na wiele niuansów związanych z prostą ogólną nazwą. W tym rozdziale przyjrzymy się matematycznemu pochodzeniu pojęcia „obserwowalność" i zbadamy, w jaki sposób praktycy rozwoju oprogramowania zaadaptowali je do opisywania cech systemów oprogramowania produkcyjnego.

Wyjaśnimy również, dlaczego adaptacja obserwowalności do użytku w systemach oprogramowania w środowiskach produkcyjnych jest konieczna. Tradycyjne praktyki debugowania wewnętrznego stanu aplikacji zostały zaprojektowane na potrzeby starszych systemów, znacznie prostszych niż te, którymi zwykle zarządzamy dzisiaj. Choć architektura systemów, platformy infrastrukturalne i oczekiwania użytkowników wciąż ewoluują, narzędzia, których używamy do analizowania tych komponentów, nie uległy zmianie. Wiele zespołów inżynierskich stosuje obecnie te same praktyki debugowania, które zostały opracowane dekady temu przy użyciu powstających wówczas narzędzi do monitorowania, mimo że systemy, którymi obecnie zarządzamy, są nieskończenie bardziej złożone. Narzędzia związane z obserwowalnością zostały zbudowane z czystej konieczności, ponieważ tradycyjne narzędzia i metody debugowania nie pozwalały na szybkie wykrywanie głęboko ukrytych i ulotnych problemów.

Ten rozdział pomoże Ci zrozumieć, co oznacza „obserwowalność", jak określić, czy system oprogramowania jest obserwowalny, dlaczego obserwowalność jest niezbędna i jak jest wykorzystywana do wykrywania problemów w sposób, który nie jest możliwy przy stosowaniu innych podejść.

Matematyczna definicja obserwowalności

Termin „obserwowalność" został wymyślony przez inżyniera Rudolfa E. Kálmána w 1960 roku. Od tego czasu pojęcie to zyskało wiele różnych znaczeń w różnych społecznościach. Zbadajmy te znaczenia, zanim przejdziemy do naszej własnej definicji dostosowanej do nowoczesnych systemów oprogramowania.

W swoim artykule z 1960 roku Kálmán wprowadził cechę, którą nazwał obserwowalnością, aby opisać matematyczne systemy sterowania[1]. W teorii sterowania (*https://w.wiki/4wHw*) obserwowalność definiuje się jako miarę tego, w jakim stopniu można wywnioskować wewnętrzne stany systemu (*https://w.wiki/55Pc*) na podstawie wiedzy o jego zewnętrznych wyjściach.

Ta definicja obserwowalności wymaga studiowania obserwowalności i sterowalności jako matematycznie powiązanych zagadnień, razem z czujnikami, równaniami algebry liniowej i metodami formalnymi. Ta tradycyjna definicja obserwowalności jest domeną inżynierów mechaników i tych, którzy zarządzają systemami fizycznymi z myślą o określonym stanie końcowym.

Jeśli szukasz podręcznika opartego na matematyce i inżynierii procesowej, trafiłeś w złe miejsce. Takie książki oczywiście istnieją, a każdy inżynier mechanik lub inżynier systemów sterowania wytłumaczy Ci (zwykle z pasją i bardzo długo), że obserwowalność ma formalne znaczenie w tradycyjnej terminologii inżynierii systemów. Jednak gdy ta sama koncepcja zostanie zaadaptowana do użytku z bardziej skomplikowanymi wirtualnymi systemami oprogramowania, daje początek zupełnie nowym sposobom interakcji z pisanym kodem i rozumienia go.

Zastosowanie obserwowalności do systemów oprogramowania

Definicję obserwowalności Kálmána można zastosować do nowoczesnych systemów oprogramowania. Aby dostosować koncepcję obserwowalności do oprogramowania, trzeba również uwzględnić dodatkowe kwestie, specyficzne dla domeny inżynierii oprogramowania. Aby aplikacja mogła być obserwowalna, musisz być w stanie wykonać następujące czynności:

- zrozumieć wewnętrzne działanie aplikacji,
- zrozumieć każdy stan systemu, w którym aplikacja mogła się znaleźć, w tym nowe stany, których nigdy wcześniej nie widziałeś i nie mogłeś przewidzieć,
- zrozumieć wewnętrzne działanie i stan systemu wyłącznie przez obserwację i interakcję z zewnętrznymi narzędziami,
- zrozumieć wewnętrzny stan *bez* pisania nowego niestandardowego kodu do jego obsługi (ponieważ oznacza to, że potrzebujesz wcześniejszej wiedzy, aby wyjaśnić stan).

Dobrym testem pomagającym ocenić, czy te warunki są spełnione, jest zadanie sobie następujących pytań:

- Czy potrafisz stale odpowiadać na otwarte pytania dotyczące wewnętrznego działania aplikacji, aby wyjaśnić wszelkie anomalie bez napotykania ślepych zaułków (na przykład sytuacji, gdy problem może dotyczyć określonej grupy mechanizmów, ale nie możesz go doprecyzować, aby potwierdzić źródło usterki)?

[1] Rudolf E. Kálmán, *On the General Theory of Control Systems* (*https://oreil.ly/u7BM4*), „IFAC Proceedings Volumes" 1, nr 1 (sierpień 1960), s. 491 – 502.

- Czy potrafisz zrozumieć, czego może doświadczać każdy konkretny użytkownik oprogramowania w danym momencie?
- Czy możesz szybko podejrzeć dowolny obraz wydajności systemu, który badasz, od zagregowanych widoków najwyższego poziomu, aż po pojedyncze konkretne żądania użytkowników, które mogą przyczyniać się do spowolnienia oprogramowania (i dane z wszystkich poziomów pośrednich)?
- Czy możesz porównać dowolne grupy żądań użytkowników w sposób pozwalający poprawnie zidentyfikować, które atrybuty są wspólne dla wszystkich użytkowników doświadczających nieoczekiwanego zachowania aplikacji?
- Czy po znalezieniu podejrzanych atrybutów w jednym żądaniu użytkownika możesz przeszukać wszystkie żądania, aby zidentyfikować podobne wzorce zachowań, by móc potwierdzić lub wykluczyć swoje podejrzenia?
- Czy możesz zidentyfikować, który użytkownik systemu generuje największe obciążenie (a zatem najbardziej spowalnia wydajność aplikacji), a także drugiego, trzeciego lub setnego użytkownika generującego największe obciążenie?
- Czy możesz zidentyfikować, który z tych użytkowników dopiero niedawno zaczął wpływać na wydajność?
- Czy jeśli 142. najwolniejszy użytkownik skarżył się na szybkość działania aplikacji, możesz wyizolować jego żądania, aby zrozumieć, dlaczego ich przetwarzanie zajmuje dużo czasu?
- Czy potrafisz znaleźć ukryte przekroczenia limitu czasu, jeśli użytkownicy skarżą się na nie, ale wykresy pokazują, że żądania z poziomu percentyla 99%, 99,9%, a nawet 99,99% są szybkie?
- Czy potrafisz odpowiedzieć na takie pytania bez konieczności przewidzenia, że pewnego dnia będziesz musiał je zadać (i w związku z tym z wyprzedzeniem konfigurujesz określone monitory w celu zagregowania niezbędnych danych)?
- Czy potrafisz odpowiedzieć na takie pytania dotyczące aplikacji, nawet jeśli nigdy wcześniej nie widziałeś ani nie debugowałeś określonego problemu?
- Czy potrafisz szybko uzyskać odpowiedzi na takie pytania, aby można było iteracyjnie zadawać nowe i jeszcze kolejne, aż dotrzesz do właściwego źródła problemów bez zakłócania toku myślenia (co zwykle oznacza, że odpowiedzi powinny być uzyskiwane w ciągu kilku sekund, a nie na przestrzeni minut)?
- Czy potrafisz odpowiedzieć na takie pytania, nawet jeśli dany problem nigdy wcześniej nie wystąpił?
- Czy wyniki debugowania często Cię zaskakują i ujawniają nowe, kłopotliwe lub dziwaczne odkrycia? A może zazwyczaj wykrywasz tylko te problemy, które podejrzewałeś, że możesz znaleźć?
- Czy potrafisz szybko (w ciągu kilku minut) wykryć każdy błąd w systemie bez względu na to, jak bardzo jest złożony lub jak głęboko kryje się w stosie oprogramowania?

Spełnienie wszystkich powyższych kryteriów okazuje się wysoko postawioną poprzeczką dla wielu organizacji zajmujących się inżynierią oprogramowania. Jeśli jesteś w stanie pokonać tę poprzeczkę, bez wątpienia rozumiesz, dlaczego obserwowalność stała się tak popularnym tematem dla zespołów inżynierów oprogramowania.

Wedle naszej definicji „obserwowalność" systemów oprogramowania jest miarą tego, jak dobrze można zrozumieć i wyjaśnić każdy stan, w jakim może znaleźć się system, bez względu na to, jak nowy lub nietypowy jest ten stan. Musisz być w stanie porównawczo debugować ten nietypowy lub nowy stan na wszystkich poziomach danych dotyczących systemu i w dowolnych kombinacjach tych poziomów, w iteracyjnych doraźnych badaniach, bez konieczności wcześniejszego definiowania lub przewidywania konkretnych potrzeb z zakresu debugowania. Jeśli możesz zrozumieć dowolny nietypowy lub nowy stan *bez konieczności pisania nowego kodu*, możesz powiedzieć, że system jest obserwowalny.

Uważamy, że takie dostosowanie tradycyjnej koncepcji obserwowalności do systemów oprogramowania jest unikalnym podejściem, z którym łączą się dodatkowe niuanse warte zbadania. W nowoczesnych systemach oprogramowania obserwowalność nie dotyczy typów danych lub danych wejściowych ani równań matematycznych. Ważne jest to, w jaki sposób ludzie wchodzą w interakcje ze złożonymi systemami i próbują je zrozumieć. Dlatego obserwowalność wymaga odkrycia interakcji między ludźmi a technologiami, aby zrozumieć, jak te złożone systemy współpracują ze sobą.

Jeśli zaakceptujesz tę definicję, pojawi się wiele dodatkowych pytań, które wymagają odpowiedzi:

- Jak zebrać potrzebne dane i połączyć je na potrzeby analiz?
- Jakie są wymagania techniczne dotyczące przetwarzania tych danych?
- Jakie umiejętności są niezbędne w zespole, aby skorzystać z tych danych?

Na te i inne pytania będziemy odpowiadać na dalszych stronach tej książki. Najpierw jednak przedstawimy dodatkowy kontekst dotyczący obserwowalności w świecie oprogramowania.

Zastosowanie obserwowalności do systemów oprogramowania ma wiele wspólnego z jej korzeniami z teorii sterowania. Jest jednak znacznie mniej matematyczne i bardziej praktyczne. Po części wynika to z faktu, że inżynieria oprogramowania jest znacznie młodszą i szybciej ewoluującą dyscypliną niż jej bardziej dojrzała poprzedniczka — inżynieria mechaniczna. Dla systemów oprogramowania produkcyjnego znacznie rzadziej można tworzyć formalne dowody. Ten brak ścisłości można uznać za zdradę w obliczu blizn, jakie zdobyliśmy w trakcie obsługi kodu oprogramowania pisanego dla środowisk produkcyjnych.

Jako inżynierowie próbujący zrozumieć, jak wypełnić lukę między teoretycznymi praktykami opartymi na testach klinicznych a skutkami tego, co się dzieje, gdy kod działa w dużej skali, nie szukaliśmy nowego terminu, definicji ani funkcji, aby opisać, jak doszliśmy do obecnych rozwiązań. To okoliczności zarządzania systemami i zespołami doprowadziły nas do odejścia od technik, które już nie działały, na przykład od monitorowania. Branża musi wyeliminować obecne luki w narzędziach i terminologii, aby uniknąć bólu i cierpienia powodowanych przez awarie i brak proaktywnych rozwiązań.

Obserwowalność ma wypełnić wspomniane luki. Złożone systemy oprogramowania produkcyjnego są skomplikowane z wielu powodów, zarówno technicznych, jak i społecznych. Aby rozwiązać związane z tym problemy, potrzebne będą techniki zarówno społeczne, jak i techniczne. Sama obserwowalność nie jest rozwiązaniem wszystkich problemów związanych z inżynierią oprogramowania. Pomaga jednak wyraźnie zobaczyć, co dzieje się we wszystkich ukrytych zakątkach oprogramowania, gdzie bez niej zwykle potykasz się po omacku, gdy próbujesz zrozumieć działanie systemu.

Wyobraź sobie, że budzisz się pewnego sobotniego poranka z wielkimi planami przygotowania brunchu dla całej rodziny. Masz w głowie wiele dań, w tym skomplikowany przepis na suflet serowy, a także listę znanych alergenów i wrażliwości pokarmowych wszystkich osób. Masz także napięty harmonogram, ponieważ babcia musi zdążyć na lotnisko przed południem. Już samo to stanowi niebanalne wyzwanie. Teraz wyobraź sobie, że nie możesz znaleźć okularów (i że jesteś tak samo krótkowzroczny jak my). Jeśli chodzi o rozwiązywanie praktycznych i wrażliwych na czas problemów w inżynierii oprogramowania, wprowadzenie obserwowalności jest cholernie dobrym punktem wyjścia.

Błędne opisy obserwowalności oprogramowania

Zanim przejdziemy dalej, musimy zająć się inną definicją „obserwowalności", promowaną zwykle przez dostawców narzędzi programistycznych typu oprogramowanie jako usługa (ang. *software-as-a-service* — SaaS). Sprzedawcy ci twierdzą, że „obserwowalność" nie ma żadnego specjalnego znaczenia, że jest to po prostu kolejny synonim „telemetrii", o znaczeniu nieodróżnialnym od „monitorowania". Zwolennicy tej definicji sprowadzają obserwowalność do kolejnego ogólnego określenia metod pomagających zrozumieć sposób działania oprogramowania. Możesz zobaczyć, jak ta grupa opisuje obserwowalność jako „trzy filary" już dostępne w rozwiązaniach, które mogą Ci sprzedać: wskaźniki, dzienniki i ślady[2].

Trudno zdecydować, co jest gorsze w tej definicji: jej redundancja (dlaczego potrzebujemy kolejnego synonimu „telemetrii"?) czy epistemiczne komplikacje (po co tworzyć listę z danymi jednego typu, „antytyp" danych w postaci chaotycznych łańcuchów znaków i na dodatek... wizualizację danych uporządkowanych według czasu?). Niezależnie od tego logiczne wady tej definicji staną się zrozumiałe, gdy zdamy sobie sprawę, że jej zwolennicy mają żywotny interes w tym, aby sprzedawać rozwiązania i sposób myślenia zbudowane wokół silosowego gromadzenia i przechowywania danych za pomocą istniejącego zestawu wskaźników, dzienników i narzędzi do śledzenia. Zwolennicy tej definicji pozwalają, aby ich modele biznesowe ograniczały sposób, w jaki myślą o przyszłych możliwościach.

Uczciwie musimy przyznać, że my, autorzy tej książki, również jesteśmy sprzedawcami w dziedzinie obserwowalności. Jednak ta książka nie została stworzona, aby sprzedać Ci nasze narzędzia. Napisaliśmy ją, aby wyjaśnić, w jaki sposób i dlaczego dostosowaliśmy oryginalną koncepcję obserwowalności do zarządzania nowoczesnymi systemami oprogramowania. Koncepcje zawarte w tej książce możesz wykorzystać niezależnie od stosowanych narzędzi, aby ćwiczyć tworzenie

[2] Czasami twierdzenia te obejmują przedziały czasowe, co pozwala dodać „dyskretne wystąpienia zmian" jako czwarty filar ogólnego synonimu „telemetrii".

produkcyjnych systemów oprogramowania z wbudowaną obserwowalnością. Obserwowalności nie osiąga się przez łączenie różnych narzędzi za pomocą technik marketingowych. Nie musisz stosować jednego konkretnego narzędzia, aby uzyskać obserwowalność w systemach oprogramowania. *Uważamy raczej, że obserwowalność wymaga ewolucji sposobu, w jaki myślimy o gromadzeniu danych potrzebnych do skutecznego debugowania systemów*. Sądzimy, że dla branży nadszedł czas na ewolucję praktyk, których używamy do zarządzania nowoczesnymi systemami oprogramowania.

Dlaczego obserwowalność jest obecnie ważna

Teraz gdy już wyjaśniliśmy, co obserwowalność oznacza, a czego nie oznacza w kontekście nowoczesnych systemów oprogramowania, porozmawiajmy o tym, dlaczego zmiana podejścia ma znaczenie. Krótko mówiąc, tradycyjne podejście polegające na korzystaniu ze wskaźników i monitorowaniu oprogramowania w celu zrozumienia jego działania jest zdecydowanie niewystarczające. Takie metody są z natury reaktywne. Być może dobrze sprawdzały się w branży w przeszłości, ale nowoczesne systemy wymagają lepszych rozwiązań.

Przez ostatnie dwie lub trzy dekady obszar między sprzętem a jego ludzkimi operatorami był zdominowany przez zestaw narzędzi i konwencji z dziedziny nazywanej przez większość osób „monitorowaniem". Praktycy odziedziczyli ten zestaw narzędzi oraz konwencji i zaakceptowali go jako najlepsze rozwiązanie pozwalające zrozumieć ten nieostry wirtualny obszar między warstwą fizyczną a kodem. Dlatego przyjęli to podejście pomimo świadomości, że w wielu sytuacjach jego nieodłączne ograniczenia będą powodować wiele bezsennych nocy spędzanych na rozwiązywaniu problemów. Mimo to nadal darzą go zaufaniem, a może nawet sympatią, ponieważ ten złodziej snu jest najlepszym, co mają.

Monitorowanie nie pozwala programistom precyzyjnie zobaczyć swoich systemów. Dlatego programiści mrużą oczy i na próżno próbują ocenić działanie oprogramowania oraz przewidzieć wszystkie niezliczone sposoby, w jakie może ono zawieść. Następnie obserwują, czyli monitorują, znane im źródła awarii. Ustalają wartości progowe z obszaru wydajności i arbitralnie określają poszczególne poziomy jako „dobre" lub „złe". Tworzą małą armię robotów do ciągłego kontrolowania wartości progowych w ich imieniu. Zapisują uzyskane dane w pulpitach nawigacyjnych. Następnie tworzą wokół tych robotów zespoły i przeprowadzają rotacje oraz eskalacje. Kiedy roboty informują, że wydajność jest zła, programiści alarmują siebie nawzajem. Następnie z biegiem czasu dbają o utrzymanie tych arbitralnych wartości progowych jak ogrodnicy: przycinają, poprawiają i marnują mnóstwo czasu na pełne szumu sygnały, które sami wyhodowali.

Czy to naprawdę najlepszy sposób?

Przez dziesięciolecia programiści i operatorzy działali w ten właśnie sposób. Monitorowanie było niepisanym standardem przez tak długi czas, że specjaliści często myślą o nim jak o *jedynym sposobie* na zrozumienie systemów i nie traktują go jak tylko *jednej z wielu możliwości*. Monitorowanie jest tak domyślnie stosowaną praktyką, że często korzysta się z niej bezrefleksyjnie. W branży przeważnie nie zastanawiamy się, czy *powinniśmy* je stosować, ale myślimy tylko, *jak* to zrobić.

Praktyka monitorowania opiera się na wielu niewypowiedzianych założeniach dotyczących systemów (te założenia szczegółowo opisujemy dalej w tym rozdziale). Jednak w miarę jak systemy ewoluują (stają się coraz bardziej abstrakcyjne i złożone, a ich podstawowe komponenty zaczynają mieć coraz mniejsze znaczenie), założenia te stają się coraz mniej prawdziwe. W miarę jak programiści i operatorzy wprowadzają nowoczesne podejścia do wdrażania systemów oprogramowania (zależności w modelu SaaS, platformy orkiestracji kontenerów, systemy rozproszone itp.), nieprawdziwość tych założeń staje się coraz bardziej oczywista.

Dlatego coraz więcej osób w bolesny sposób zderza się z nieodłącznymi ograniczeniami monitorowania i zdaje sobie sprawę, że oparte na nim podejścia nie sprawdzają się w obecnym nowoczesnym świecie. Tradycyjne praktyki monitorowania są katastrofalnie nieskuteczne, jeśli chodzi o możliwość zrozumienia systemów. Założenia dotyczące wskaźników i monitorowania nie są obecnie spełnione. Aby zrozumieć, dlaczego tak się dzieje, warto przyjrzeć się ich historii i pierwotnemu kontekstowi ich stosowania.

Dlaczego wskaźniki i monitorowanie nie wystarczają?

W 1988 roku, wraz z protokołem Simple Network Management Protocol (SNMPv1 zdefiniowanym w dokumencie RFC 1157), narodził się podstawowy komponent monitorowania: wskaźnik. **Wskaźnik** to pojedyncza liczba z opcjonalnie dołączonymi tagami umożliwiającymi grupowanie i wyszukiwanie takich danych. Wskaźniki są z natury jednorazowe i tanie. Zajmują przewidywalną ilość pamięci. Można je łatwo agregować w regularnych odstępach czasu. Dlatego wskaźnik stał się podstawową jednostką dla jednej lub dwóch generacji telemetrii. Reprezentuje dane, które zbieramy ze zdalnych punktów końcowych w celu automatycznego przesyłania do systemów monitorowania.

Na podstawie wskaźników opracowano wiele wyrafinowanych rozwiązań: bazy danych dla szeregów czasowych (ang. *time-series databases* — TSDB), analizy statystyczne, biblioteki wykresów, wymyślne pulpity nawigacyjne, dyżury, zespoły operacyjne, zasady eskalacji, a także mnóstwo sposobów na przetwarzanie i reagowanie na to, jakie informacje przekazuje mała armia robotów.

Istnieje jednak górna granica złożoności systemów, które można zrozumieć za pomocą wskaźników i narzędzi monitorujących. Po przekroczeniu tej granicy następuje gwałtowna zmiana. To, co działało wystarczająco dobrze w zeszłym miesiącu, nagle przestaje się sprawdzać. Zaczynasz wracać do niskopoziomowych poleceń, takich jak `strace`, `tcpdump` i setki instrukcji `print`, aby odpowiedzieć na pytania dotyczące zachowania systemu.

Trudno jest dokładnie obliczyć, kiedy zostanie osiągnięty ten punkt krytyczny. W końcu sama liczba możliwych stanów, w jakie może przejść system, przekroczy zdolność zespołu do dopasowywania wzorców na podstawie wcześniejszych awarii. Trzeba zrozumieć zbyt wiele nowych, nieznanych stanów. Zespół nie może już ustalić, jakie pulpity nawigacyjne należy utworzyć, aby wyświetlać niezliczone symptomy awarii.

Narzędzia monitorujące i oparte na wskaźnikach zostały zbudowane zgodnie z określonymi założeniami dotyczącymi architektury i organizacji. W praktyce te założenia służyły jako ograniczenie złożoności. Założenia te są zwykle niewidoczne, dopóki ich nie przekroczysz. W tym momencie przestają być ukryte i upośledzają zdolność do zrozumienia tego, co się dzieje. Oto niektóre z takich założeń:

- Aplikacja ma budowę monolityczną.
- Istnieje jeden stanowy magazyn danych („baza danych”), który samodzielnie obsługujesz.
- Dostępnych jest wiele niskopoziomowych wskaźników systemowych (na przykład używana pamięć fizyczna lub średnie obciążenie procesora).
- Aplikacja działa w kontenerach, maszynach wirtualnych lub fizycznych maszynach, które kontrolujesz.
- Wskaźniki systemowe i wskaźniki instrumentacji są głównym źródłem informacji używanych do debugowania kodu.
- Masz stosunkowo statyczny zestaw długo działających węzłów, kontenerów lub hostów do monitorowania.
- Inżynierowie badają systemy pod kątem problemów dopiero po ich wystąpieniu.
- Pulpity nawigacyjne i telemetria istnieją po to, aby zaspokoić potrzeby inżynierów operacyjnych.
- Systemy monitorowania badają aplikacje „czarnoskrzynkowe” w podobny sposób jak aplikacje lokalne.
- Monitorowanie koncentruje się na czasie pracy i zapobieganiu awariom.
- Badanie korelacji odbywa się w ograniczonej (lub niewielkiej) liczbie wymiarów.

W zestawieniu tych założeń z realiami współczesnych systemów staje się jasne, że tradycyjne podejścia do monitorowania nie sprawdzają się w kilku obszarach. Rzeczywistość nowoczesnych systemów wygląda następująco:

- Aplikacje obejmują wiele usług.
- Stosowane jest niejednorodne utrwalanie danych (czyli wiele baz danych i systemów pamięci masowej).
- Infrastruktura jest niezwykle dynamiczna, a zasoby są nieustannie dodawane i usuwane.
- Używanych jest wiele rozproszonych i luźno powiązanych usług, z których wiele nie pozostaje bezpośrednio pod kontrolą użytkownika.
- Inżynierowie aktywnie sprawdzają wpływ zmian w kodzie produkcyjnym, aby wcześnie wychwycić drobne problemy, zanim wpłyną one na użytkownika.
- Automatyczna instrumentacja jest niewystarczająca do zrozumienia, co dzieje się w złożonych systemach.
- Inżynierowie oprogramowania są właścicielami własnego kodu produkcyjnego i zachęca się ich do proaktywnej instrumentacji kodu oraz sprawdzania wpływu nowych zmian na wydajność już na etapie ich wdrażania.
- Zapewnianie niezawodności polega głównie na ustaleniu, jak uzyskać tolerancję na stałą i ciągłą degradację, a jednocześnie budować odporność na awarie mające wpływ na użytkownika dzięki wykorzystaniu takich mechanizmów jak budżet błędów, jakość usług i doświadczenia użytkownika.
- Badanie korelacji odbywa się w praktycznie nieograniczonej liczbie wymiarów.

Ostatni punkt jest ważny, ponieważ opisuje rozłam między granicami wiedzy o korelacjach, jaką może posiadać jeden człowiek, a realiami nowoczesnych architektur systemowych. W procesie odkrywania korelacji leżących u podstaw problemów z wydajnością trzeba uwzględnić tak wiele wymiarów, że żaden ludzki mózg, a w praktyce nawet żaden schemat, nie jest w stanie ich pomieścić.

W obserwowalności porównywanie danych o wielu wymiarach i wysokiej kardynalności jest kluczowym elementem umożliwiającym odkrywanie problemów ukrytych w złożonych architekturach systemu.

Debugowanie z wykorzystaniem wskaźników a obserwowalność

Gdy złożoność systemu przekroczy poziom krytyczny, nie da się już zmieścić modelu systemu w mentalnej pamięci podręcznej. Zanim zdążysz przeanalizować różne komponenty, Twój model mentalny prawdopodobnie stanie się już nieaktualny.

Jeśli jesteś inżynierem, prawdopodobnie jesteś przyzwyczajony do debugowania z wykorzystaniem intuicji. Aby dotrzeć do źródła problemu, prawdopodobnie kierujesz się przeczuciem lub korzystasz z ulotnego wspomnienia awarii, która miała miejsce dawno temu. Jednak umiejętności, które dobrze służyły Ci w przeszłości, w tym świecie nie mają już zastosowania. Podejście intuicyjne działa tylko tak długo, jak długo większość napotykanych problemów to odmiany kilku tych samych przewidywalnych scenariuszy, z którymi zetknąłeś się w przeszłości[3].

Podobnie oparte na wskaźnikach podejście do monitorowania wymaga napotkania znanych symptomów awarii w przeszłości. Monitorowanie pomaga wykryć, kiedy wskaźniki w systemie są za wysokie lub za niskie względem przewidywalnych wartości progowych, których naruszenie ktoś wcześniej uznał za anomalię. *Ale co zrobić, gdy nie wiadomo, czy określony rodzaj anomalii może w ogóle wystąpić?*

W przeszłości większość problemów napotykanych przez inżynierów oprogramowania stanowiły odmiany dość przewidywalnych symptomów awarii. Być może nie było wiadomo, że oprogramowanie może zawieść w określony sposób, ale po przeanalizowaniu sytuacji i istotnych komponentów odkrycie nowego błędu lub symptomu awarii nie wymagało skoku logicznego. Większość twórców oprogramowania rzadko spotyka się z naprawdę nieprzewidywalnymi skokami logicznymi, ponieważ zazwyczaj nie mają do czynienia z typem złożoności sprawiającym, że takie skoki są powszechne (do tej pory większość złożoności znanej programistom była zawarta w monolitycznych aplikacjach).

> *Każda aplikacja ma naturalnie występujący poziom nieredukowalnej złożoności. Jedyne pytanie brzmi: kto będzie musiał sobie z nią poradzić — użytkownik, twórca aplikacji czy twórca platformy?*
>
> — Larry Tesler

[3] Bardziej szczegółową analizę znajdziesz w artykule z bloga Pete'a Hodgsona, *Why Intuitive Troubleshooting Has Stopped Working for You* (*https://oreil.ly/JXx0c*).

Architektury nowoczesnych systemów rozproszonych notorycznie zawodzą w nietypowe sposoby, których nikt nie jest w stanie przewidzieć i których nikt wcześniej nie doświadczył. Zdarza się to na tyle często, że powstała cała lista fałszywych założeń (*https://w.wiki/56wa*) często przyjmowanych przez programistów dopiero wkraczających w świat przetwarzania rozproszonego. Nowoczesne systemy rozproszone są również udostępniane twórcom aplikacji jako *abstrakcyjne platformy infrastrukturalne*. Jako użytkownicy tych platform twórcy aplikacji są obecnie zmuszeni do radzenia sobie z naturalnie występującym poziomem nieredukowalnej złożoności, która trafia prosto na ich biurka.

Wcześniej ukryta złożoność podprocedur z kodu aplikacji, które współdziałały ze sobą w ukrytej pamięci wewnętrznej o dostępie swobodnym w jednej maszynie fizycznej, przyjmuje obecnie widoczną postać żądań usług przekazywanych między hostami. Ta nowo ujawniona złożoność przenika następnie przez wiele usług, a żądania wielokrotnie pokonują nieprzewidywalną sieć węzłów w trakcie wykonywania pojedynczej funkcji. Kiedy nowoczesne architektury sprawiły, że wygodniejsze stało się rozkładanie monolitów na mikrousługi, inżynierowie oprogramowania stracili możliwość analizowania własnego kodu za pomocą tradycyjnych debugerów. Jednocześnie dostępne narzędzia jeszcze nie radziły sobie z tą olbrzymią zmianą.

Krótko mówiąc, rozsadziliśmy monolit. Teraz każde żądanie wielokrotnie przepływa przez sieć, a każdy programista musi lepiej znać się na systemach i operacjach, aby wykonywać codzienną pracę.

Przykładami tej olbrzymiej zmiany mogą być trend w kierunku konteneryzacji, rozwój platform orkiestracji kontenerów, przechodzenie na mikrousługi, powszechne stosowanie niejednorodnego utrwalania danych, wprowadzanie siatki usług, popularność tymczasowych instancji z automatycznym skalowaniem, przetwarzanie bezserwerowe, funkcje lambda i wszelkie inne niezliczone aplikacje w modelu SaaS znajdujące się obecnie w typowym zestawie narzędzi programisty. Połączenie tych różnych narzędzi w architekturę nowoczesnego systemu sprawia, że żądanie może wykonać od 20 do 30 przeskoków po tym, jak dotrze do obrzeży kontrolowanej przez Ciebie sieci (tę wartość często trzeba pomnożyć przez dwa, jeśli żądanie obejmuje zapytania do bazy danych).

W nowoczesnych systemach natywnych dla chmury najtrudniejszą rzeczą w debugowaniu nie jest już zrozumienie, jak działa kod, ale *znalezienie miejsca w systemie*, w którym znajduje się kod będący źródłem problemu. Życzymy powodzenia w szukaniu wolno działających węzłów lub usług za pomocą pulpitów nawigacyjnych lub map usług. Jest to bardzo trudne, ponieważ rozproszone żądania w nowoczesnych systemach często się zapętlają. Znalezienie wąskich gardeł obniżających wydajność takich systemów jest niezwykle trudne. Gdy coś działa wolno, *wszystko działa wolno*. Co więcej, ponieważ systemy natywne dla chmury zazwyczaj działają jako platformy, kod może znajdować się w części systemu, której zespół nawet nie kontroluje.

We współczesnym świecie debugowanie z wykorzystaniem wskaźników wymaga połączenia dziesiątek niepowiązanych ze sobą wartości, które zostały zarejestrowane w trakcie wykonywania jednego konkretnego żądania z użyciem dowolnej liczby usług lub maszyn. Dopiero wtedy można wywnioskować, co mogło się wydarzyć w różnych przeskokach potrzebnych do wykonania danego żądania. Przydatność tych dziesiątek wskazówek zależy od tego, czy ktoś był w stanie z wyprzedzeniem przewidzieć, czy dany pomiar znajdował się powyżej lub poniżej wartości progowej pozwalającej określić, że konkretna operacja przyczyniła się do wystąpienia nieznanego i nigdy wcześniej niespotkanego symptomu awarii.

Natomiast debugowanie z wykorzystaniem obserwowalności zaczyna się od zupełnie innego punktu — szczegółowego kontekstu tego, co się działo w trakcie wykonywania danej operacji. Debugowanie z użyciem obserwowalności polega na zachowaniu jak największej ilości kontekstu związanego z danym żądaniem, tak aby można było zrekonstruować środowisko i okoliczności, które wywołały błąd prowadzący do nowego symptomu awarii. Monitorowanie dotyczy „znanych niewiadomych", natomiast obserwowalność służy do badania „nieznanych niewiadomych".

Znaczenie kardynalności

W kontekście baz danych **kardynalność** związana jest z unikatowością wartości danych zawartych w zbiorze. **Niska kardynalność** oznacza, że kolumna zawiera wiele powtarzających się wartości. **Wysoka kardynalność** oznacza, że kolumna zawiera duży procent unikatowych wartości. Kolumna zawierająca pojedynczą wartość zawsze będzie miała najniższą możliwą kardynalność, a kolumna z unikalnymi identyfikatorami zawsze będzie miała najwyższą możliwą kardynalność.

Na przykład w zbiorze stu milionów rekordów użytkowników można założyć, że każda kolumna z identyfikatorem UUID będzie miała najwyższą możliwą kardynalność. Innym przykładem wysokiej kardynalności mogą być podpisy oparte na kluczach publicznych. Imię i nazwisko mają wysoką kardynalność, choć niższą niż identyfikatory UUID, ponieważ niektóre imiona i nazwiska się powtarzają. Pole takie jak Płeć miałoby niską kardynalność, gdyby schemat bazy danych został utworzony 50 lat temu, ale z uwagi na obecne rozumienie płci dziś sytuacja może wyglądać inaczej. Pole takie jak Gatunek miałoby najniższą możliwą kardynalność, jeśli przyjąć założenie, że wszyscy użytkownicy są ludźmi.

Kardynalność ma znaczenie dla obserwowalności, ponieważ *informacje o wysokiej kardynalności są prawie zawsze najbardziej przydatne* do identyfikowania danych przydatnych do debugowania lub zrozumienia systemu. Rozważ przydatność sortowania danych według pól takich jak identyfikatory użytkowników, identyfikatory koszyków zakupów, identyfikatory żądań lub inne niezliczone identyfikatory, na przykład identyfikatory instancji, kontenerów, hostów, zakresów, numery kompilacji i tak dalej. Umożliwienie przesyłania zapytań o unikalne identyfikatory to najlepszy sposób na wyszukiwanie pojedynczych igieł w stogu siana. Zawsze można przeprowadzić skalowanie wartości o wysokiej kardynalności na dane o niższej kardynalności (na przykład pogrupować nazwiska według początkowych liter), ale nigdy nie można wykonać odwrotnej operacji.

Niestety, systemy narzędzi oparte na wskaźnikach radzą sobie tylko z wymiarami o niskiej kardynalności i w rozsądnej skali. Nawet jeśli masz tylko setki hostów do porównania, w systemach opartych na wskaźnikach nie możesz użyć nazwy hosta jako tagu identyfikującego bez przekroczenia limitów kardynalności w przestrzeni kluczy.

Te nieodłączne ograniczenia skutkują nieplanowanymi restrykcjami, jeśli chodzi o sposoby badania danych. Podczas debugowania za pomocą wskaźników dla każdego pytania na temat danych, które możesz chcieć zadać, musisz zdecydować — *z wyprzedzeniem*, przed wystąpieniem błędu — jakie informacje będą potrzebne. Tylko wtedy daną wartość można zarejestrować w trakcie zapisywania wskaźników.

Ma to dwie ważne implikacje. Po pierwsze, jeśli w trakcie analizy zdecydujesz, że musisz zadać dodatkowe pytanie, aby odkryć źródło potencjalnego problemu, nie da się tego zrobić po fakcie. Musisz najpierw skonfigurować wskaźniki, które pomogą odpowiedzieć na dane pytanie, a następnie poczekać, aż problem się powtórzy. Po drugie odpowiedź na to dodatkowe pytanie wymaga kolejnego zestawu wskaźników, a większość dostawców narzędzi opartych na wskaźnikach pobiera opłaty za rejestrowanie kolejnych danych. Koszt wzrasta więc liniowo wraz z każdym nowym sposobem, w jaki decydujesz się przeszukiwać dane, by znaleźć ukryte problemy, których nie zdołałeś wcześniej przewidzieć.

Znaczenie liczby wymiarów

O ile *kardynalność* dotyczy unikatowości wartości w danych, o tyle **liczba wymiarów** związana jest z liczbą kluczy w tych danych. W obserwowalnych systemach dane telemetryczne są generowane jako zdarzenia o dowolnie szerokiej strukturze (zobacz rozdział 8.). Zdarzenia te są określane jako „szerokie", ponieważ mogą i powinny zawierać setki, a nawet tysiące par klucz-wartość (lub wymiarów). Im szersze zdarzenie, tym bogatszy kontekst uchwycony w momencie wystąpienia zdarzenia, a tym samym w trakcie późniejszego debugowania można dowiedzieć się więcej na temat tego, co się stało.

Wyobraź sobie, że masz schemat zdarzeń, który definiuje sześć wymiarów o wysokiej kardynalności zapisywanych dla każdego zdarzenia: `time`, `app`, `host`, `user`, `endpoint` i `status`. Dzięki tym sześciu wymiarom można tworzyć zapytania na potrzeby analizy dowolnej kombinacji wymiarów w celu wykrycia istotnych wzorców, które mogą przyczyniać się do anomalii. Możesz na przykład pobrać „wszystkie błędy 502, które wystąpiły w ciągu ostatniej pół godziny w hoście `foo`" lub „wszystkie błędy 403 wygenerowane przez żądania do punktu końcowego `/export` wykonane przez użytkownika `bar`", lub „wszystkie przekroczenia limitu czasu, które wystąpiły wskutek żądań wysłanych do punktu końcowego `/payments` przez aplikację `baz`, wraz z nazwą źródłowego hosta tych żądań".

Dzięki zaledwie sześciu podstawowym wymiarom można zbadać przydatny zestaw warunków, aby określić, co może się dziać w systemie z aplikacją. Teraz wyobraź sobie, że zamiast tylko sześciu wymiarów masz dostęp do setek lub tysięcy wymiarów reprezentujących dowolne szczegóły, wartości, liczniki lub ciągi, które mogą okazać się przydatne do debugowania w jakimś momencie w przyszłości. Oto przykładowa lista wymiarów:

```
app.api_key
app.batch
app.batch_num_data_sets
app.batch_total_data_points
app.dataset.id
app.dataset.name
app.dataset.partitions
app.dataset.slug
app.event_handler
app.raw_size_bytes
app.sample_rate
app.team.id
…
response.content_encoding
response.content_type
```

```
response.status_code
service_name
trace.span_id
trace.trace_id
```

Mając do dyspozycji znacznie więcej wymiarów, można badać zdarzenia w celu wykrywania złożonych korelacji między dowolnymi grupami żądań z usługi (zobacz rozdział 8.). Im bardziej wielowymiarowe są dane, tym większe prawdopodobieństwo znalezienia ukrytych lub ulotnych wzorców w zachowaniu aplikacji. W nowoczesnych systemach, w których liczba różnych awarii, jakie mogą wystąpić, jest praktycznie nieograniczona, zapisywanie tylko kilku podstawowych wymiarów w danych telemetrycznych jest niewystarczające.

Musisz zebrać bardzo bogate szczegóły dotyczące wszystkiego, co dzieje się na styku użytkowników, kodu i systemów. Dane o dużej liczbie wymiarów zapewniają szerszy kontekst obrazujący, jak przebiegają interakcje między tymi jednostkami. W późniejszych rozdziałach omówimy, w jaki sposób dane o wysokiej liczbie wymiarów (często o wysokiej kardynalności) są analizowane w celu ujawnienia, gdzie w systemie występują interesujące Cię problemy i jakie są ich przyczyny.

Debugowanie przy zapewnionej obserwowalności

Zamiast ograniczać kardynalność i liczbę wymiarów danych telemetrycznych, narzędzia do zapewniania obserwowalności zachęcają deweloperów do gromadzenia bogatych danych tego rodzaju dla każdego zdarzenia, które może wystąpić. Pozwala to uchwycić pełny kontekst każdego żądania i zachować go do ewentualnego wykorzystania w przyszłości. Narzędzia do zapewniania obserwowalności są specjalnie zaprojektowane, aby umożliwiać wykonywanie zapytań o dane o wysokiej kardynalności i dużej liczbie wymiarów.

Dlatego też w celu debugowania można przeszukiwać dane dotyczące zdarzeń na dowolną liczbę rozmaitych sposobów. Dzięki obserwowalności można iteracyjnie badać interesujące warunki, eksplorować dane i sprawdzać, co mogą ujawnić na temat stanu systemu. Zadajesz pytanie, którego nie musiałeś wcześniej przewidzieć, aby znaleźć odpowiedzi lub wskazówki, które doprowadzą Cię do kolejnego pytania, a później następnego i jeszcze innego. Powtarzasz ten schemat raz za razem, aż znajdziesz przysłowiową igłę w stogu siana. Kluczową cechą obserwowalnych systemów jest możliwość eksploracji systemu w sposób otwarty.

Miarą możliwości eksploracji systemu jest to, jak precyzyjnie można zadać dowolne pytanie i sprawdzić potrzebny mu stan wewnętrzny. **Możliwość eksploracji** oznacza, że możesz iteracyjnie zbadać i ostatecznie zrozumieć dowolny stan, w jakim znalazł się system — nawet jeśli nigdy wcześniej nie widziałeś danego stanu — bez konieczności wcześniejszego przewidywania, jakie stany mogą wystąpić. Warto powtórzyć, że obserwowalność oznacza, iż możesz zrozumieć i wyjaśnić każdy stan (bez względu na to, jak nowy lub nietypowy on jest), w jaki może przejść system bez dodawania nowego kodu.

Monitorowanie działało tak dobrze przez tak długi czas, ponieważ systemy były na tyle proste, że inżynierowie mogli dokładnie zrozumieć, gdzie należy szukać problemów i jak te problemy mogą się objawiać. Na przykład stosunkowo łatwo jest się domyślić, że gdy gniazda się zapełnią, procesor zostanie przeciążony, a rozwiązaniem jest wtedy zwiększenie przepustowości przez

przeskalowanie węzłów aplikacji lub przez dostrojenie bazy danych itd. Inżynierowie zwykle mogli z góry przewidzieć większość możliwych stanów związanych z awariami, a resztę odkryć w bardziej bolesny sposób, gdy aplikacje działały już w środowisku produkcyjnym.

Monitorowanie prowadzi jednak do reaktywnego podejścia do zarządzania systemem. Możesz wychwycić warunki wystąpienia awarii, jeśli je przewidziałeś i wiedziałeś, że należy je sprawdzać. Jeżeli wiesz, że możesz się czegoś spodziewać, sprawdzasz to. Jeśli jednak chodzi o stany, o których nie wiesz, że należy ich szukać, musisz je najpierw napotkać, poradzić sobie z nieprzyjemną niespodzianką, zbadać je najlepiej, jak potrafisz, i być może wejść w ślepy zaułek, dlatego konieczne może być wielokrotne napotkanie tego samego stanu przed jego prawidłowym zdiagnozowaniem. Dopiero po tym wszystkim możesz opracować odpowiednie testy. W tym modelu w inżynierach narasta silna niechęć do sytuacji, które mogą powodować nieprzewidywalne awarie. Po części właśnie dlatego niektóre zespoły boją się wdrażać nowy kod (więcej na ten temat dowiesz się później).

Jedna subtelna dodatkowa uwaga — problemy ze sprzętem lub infrastrukturą są proste w porównaniu z tymi wywoływanymi przez kod lub użytkowników. Przejście od „większość moich problemów to awarie komponentów" do „większość moich pytań ma związek z zachowaniem użytkownika lub trudnymi do wykrycia błędami w kodzie i interakcjami" jest powodem, dla którego nawet firmy z monolitycznymi aplikacjami i prostymi architekturami mogą dążyć do zastąpienia monitorowania obserwowalnością.

Obserwowalność jest dostosowana do nowoczesnych systemów

System oprogramowania produkcyjnego jest obserwowalny w takim zakresie, w jakim można zrozumieć nowe wewnętrzne stany systemu bez konieczności arbitralnych domysłów, przewidywania symptomów awarii z wyprzedzeniem lub tworzenia nowego kodu, aby zrozumieć dany stan. W ten sposób rozszerzamy koncepcję obserwowalności z teorii sterowania na dziedzinę inżynierii oprogramowania.

W inżynierii oprogramowania obserwowalność zapewnia korzyści także osobom korzystającym z bardziej tradycyjnych architektur lub systemów monolitycznych. *Zawsze* pomocna jest możliwość prześledzenia kodu i sprawdzenia, na co przeznaczany jest czas przetwarzania, lub odtworzenia zachowania oprogramowania z perspektywy użytkownika. Te możliwości z pewnością mogą uchronić zespoły przed koniecznością odkrywania nieprzewidywalnych symptomów awarii w środowisku produkcyjnym i to bez względu na używaną architekturę. Ale to właśnie w nowoczesnych systemach rozproszonych „skalpel i reflektor" zapewniane przez narzędzia z obszaru obserwowalności stają się bezwzględnie niezbędne.

W systemach rozproszonych stosunek dość przewidywalnych symptomów awarii do nowych i nigdy wcześniej niewidzianych jest mocno zaburzony na rzecz tych nietypowych i nieprzewidywalnych. Te nieprzewidywalne symptomy awarii pojawiają się tak często i powtarzają na tyle rzadko, że większość zespołów nie potrafi skonfigurować odpowiednich i wystarczająco adekwatnych pulpitów monitorujących, aby łatwo zobrazować stan zespołom inżynieryjnym odpowiedzialnym za zapewnienie ciągłego czasu pracy, niezawodności i akceptowalnej wydajności aplikacji produkcyjnych.

Napisaliśmy tę książkę z myślą o tego typu nowoczesnych systemach. W każdym systemie składającym się z wielu komponentów, które są ze sobą luźno powiązane, dynamiczne i trudne do zrozumienia, korzystne będzie wprowadzenie obserwowalności zamiast tradycyjnych metod zarządzania. Jeśli zarządzasz produkcyjnymi systemami oprogramowania, które pasują do tego opisu, z tej książki dowiesz się, co obserwowalność może oznaczać dla Ciebie, Twojego zespołu, Twoich klientów i Twojej firmy. Skupiamy się również na aspektach ludzkich niezbędnych do rozwinięcia praktyki obserwowalności w najważniejszych obszarach procesów inżynieryjnych.

Podsumowanie

Chociaż pojęcie *obserwowalność* jest definiowane od dziesięcioleci, jego zastosowanie w systemach oprogramowania jest nowym podejściem, z czym wiążą się nowe zagadnienia i cechy. W porównaniu z ich prostszymi wczesnymi odpowiednikami nowoczesne systemy wprowadzają tyle dodatkowej złożoności, że przewidywanie, wykrywanie i eliminowanie awarii stało się trudniejsze niż kiedykolwiek wcześniej.

Aby ograniczyć skutki tej złożoności, zespoły inżynierskie muszą teraz być w stanie stale gromadzić dane telemetryczne w elastyczny sposób, który pozwala diagnozować problemy bez konieczności przewidywania, w jaki sposób mogą wystąpić awarie. Obserwowalność umożliwia inżynierom precyzyjną analizę danych telemetrycznych w elastyczny sposób, który pozwala dotrzeć do źródła wszelkich występujących problemów w sposób niemożliwy przy użyciu innych metod.

Obserwowalność często błędnie uznaje się za osiągniętą, gdy dostępne są „trzy filary" różnych typów danych telemetrycznych. Nie jesteśmy fanami tego modelu, jeśli jednak musimy wymienić trzy filary obserwowalności, to powinny to być narzędzia, które zapewniają wysoką kardynalność, dużą liczbę wymiarów i możliwość eksploracji. W następnym rozdziale zbadamy, jak obserwowalność różni się od tradycyjnego podejścia do monitorowania systemów.

To ważna lektura dla każdego, kto chce zrozumieć obserwowalność systemów oprogramowania!

Alex Hidalgo, autor książki *Implementing Service Level Objectives*

Obserwowalność jest często mylnie uznawana za monitorowanie systemu. Tymczasem system jest obserwowalny, jeśli dzięki pochodzącym z niego danym można zrozumieć, jak on działa, jakie występują w nim problemy i jak wpływają one na jego działanie. Cechę tę można z powodzeniem wykorzystać w produkcyjnych systemach oprogramowania.

Wprowadzenie obserwowalności do systemów jest wyzwaniem technicznym i kulturowym. Dzięki tej praktycznej książce zrozumiesz wartość obserwowalnych systemów i nauczysz się praktykować programowanie sterowane obserwowalnością. Przekonasz się, że dzięki jej wdrożeniu zespoły mogą szybko i bez obaw dostarczać kod, identyfikować wartości odstające i nietypowe zachowania, a ponadto lepiej zrozumieją doświadczenia użytkownika. Znajdziesz tu szczegółowe wyjaśnienia, co jest potrzebne do uzyskania wysokiej obserwowalności, a także szereg wskazówek, jak ulepszyć istniejące rozwiąznia i pomyślnie dokonać migracji ze starszych narzędzi, takich jak wskaźniki, monitorowanie i zarządzanie dziennikami. Dowiesz się również, jaki wpływ ma obserwowaność systemu na kulturę organizacji — i odwrotnie.

W książce:

- stosowanie obserwowalności do zarządzania oprogramowaniem w dużej skali
- obserwowalność w procesie dostarczania złożonych aplikacji i systemów natywnych dla chmury
- wpływ obserwowalności na cały cykl życia oprogramowania
- stosowanie obserwowalności w połączeniu z poziomami SLO
- instrumentacja kodu
- debugowanie nieuchwytnych problemów

Charity Majors jest współzałożycielką i CTO firmy Honeycomb. Wcześniej pracowała między innymi w Parse, Facebooku i Linden Lab.

Liz Fong-Jones jest rzeczniczką deweloperów i inżynierem niezawodności z ponad 17-letnim doświadczeniem. Obecnie zajmuje się inżynierią niezawodności i obserwowalnością.

George Miranda był inżynierem systemów, aktualnie zajmuje się marketingiem produktów. Wcześniej przez ponad 15 lat budował systemy rozproszone w branży finansowej i gier wideo.

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 978-83-289-1198-7
9 788328 911987
Cena: 69,00 zł