O'REILLY®

Wydanie II

Projektowanie systemów rozproszonych

Wzorce i paradygmaty dla skalowalnych, niezawodnych usług z wykorzystaniem Kubernetesa

Helion

Brendan Burns

Tytuł oryginału: Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Systems Using Kubernetes, 2nd Edition

Tłumaczenie: Anna Mizerska, z wykorzystaniem fragmentów poprzedniego wydania w przekładzie Lecha Lachowskiego

ISBN: 978-83-289-3147-3

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

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
helion.pl/user/opinie/prsyr2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

Część II. Wzorce jednowęzłowe

Część III. Wzorce serwowania usług

Część IV. Wzorce przetwarzania wsadowego

Część V. Pojęcia uniwersalne

ROZDZIAŁ 14.

Wzorce monitorowania i obserwowalności

Jedną z najważniejszych różnic między aplikacjami klienckimi a systemami rozproszonymi jest to, że systemy rozproszone zazwyczaj implementują usługi. Usługi te są zawsze włączone i dostępne dla użytkowników na całym świecie we wszystkich strefach czasowych. Ze względu na całodobowy (nieprzerwany) charakter tych systemów monitorowanie i obserwowalność stają się kluczowe dla budowy niezawodnych systemów. Aby zapewnić niezawodność, musisz zauważyć problem, zanim zrobi to klient, a żeby rozwiązać napotkane problemy, musisz zrozumieć, jak działa system. W tym rozdziale skupimy się na najlepszych praktykach w zakresie monitorowania i obserwowalności.

Podstawy monitorowania i obserwowalności

Zanim przejdziemy do szczegółów wdrażania monitorowania i obserwowalności, warto zrozumieć podstawowy zestaw pojęć, które wiążą się z każdym monitorowanym i obserwowalnym rozwiązaniem.

W każdym systemie istnieją cztery kluczowe pojęcia, które składają się na rozwiązania:

- zapisy w logach,
- wskaźniki,
- alarmowanie,
- śledzenie.

Przyjrzyjmy się dokładnie każdemu z tych pojęć, by je dobrze zrozumieć.

Zapewne każdy, kto zbudował choćby najmniejszy system, zaimplementował zapis w logach, nawet jeśli nie zdaje sobie z tego sprawy. Najprostszą wersją rejestrowania zdarzeń jest skromna instrukcja `printf`. Oczywiście istnieje wiele bardziej zaawansowanych sposobów zapisu w logach, ale ostatecznie wszystkie one służą temu samemu celowi co ta prosta instrukcja wyświetlania.

Mianowicie pokazują nam, że konkretne miejsce w naszym kodzie zostało wykonane, i rejestrują dane związane z tym miejscem. Zapisywane dane pomagają nam zrozumieć konkretne działania kodu.

W przeciwieństwie do zapisywania w logach wskaźniki lub monitorowanie odnoszą się do danych zbiorczych z wielu żądań. Przykłady wskaźników to liczba wywołań danej funkcji w ostatniej minucie czy średni czas spędzony w tej funkcji we wszystkich tych wywołaniach. Logi dostarczają informacje, które pozwalają zrozumieć konkretne wykonanie usługi, natomiast wskaźniki dostarczają całościowy obraz tego, jak usługa działa ogólnie.

Celem logów i wskaźników jest pomoc w zrozumieniu systemów, gdy je analizujemy lub wiemy, że istnieje problem. Moglibyśmy cały czas patrzeć na wykresy wskaźników i interfejsy rejestrów, lecz oczywiście jest to niemożliwe, gdyż spędzamy wiele czasu w innych środowiskach. Dlatego potrzebujemy systemu alarmowania, który wykryje problemy i powiadomi nas, że musimy przerwać swoje zajęcia i spojrzeć na logi lub wskaźniki. **Alerty** to stosowane do logów lub wskaźników zasady, które wyzwalają zdarzenie. Zazwyczaj to zdarzenie uruchamia powiadomienie mające na celu zwrócenie naszej uwagi na problem (lub potencjalny problem) w systemie. Tworzenie wysokiej jakości alertów jest kluczowym aspektem budowania niezawodnego systemu.

Ostatni komponent systemu rozproszonego odnosi się do jego rozproszonej części. Jednym z najtrudniejszych problemów przy budowie systemu rozproszonego jest zrozumienie, jak wszystkie elementy współpracują, a jeszcze trudniejsze jest zrozumienie, dlaczego mogą zawodzić w określonych okolicznościach. Na pierwszy rzut oka każde żądanie do mikrousługi jest unikatowe. Jednak w rzeczywistości, gdy patrzymy z szerszej perspektywy aplikacji, często istnieje większe żądanie, które jest obsługiwane przez łańcuch wywołań przez wszystkie mikrousługi. **Śledzenie** to proces odtwarzania tego kontekstu, dzięki czemu zamiast widzieć pojedyncze, unikalne żądania w każdej mikrousłudze, możemy zrozumieć, że wszystkie te indywidualne elementy są częścią szerszego

żądania użytkownika. Ta globalna perspektywa umożliwia zrozumienie (na przykład), dlaczego żądanie konkretnego użytkownika działa wolno lub wcale nie działa. Śledzenie łączy wszystkie elementy systemu rozproszonego i rekonstruuje perspektywę użytkownika.

W kolejnych podrozdziałach przyjrzymy się każdemu z tych komponentów i zobaczymy, jak można je wykorzystać do budowy niezawodnego systemu.

Rejestrowanie w dzienniku

Kiedy już opanujesz, jak zapisywać informacje w dziennikach (logach), jedną z pierwszych rzeczy, które zauważysz w systemie rozproszonym, jest ogromna ilość danych, które są rejestrowane. Objętość logów, czyli „spam logowy", może często stanowić znaczną trudność w zrozumieniu problemu, a także w zakresie kosztów ich przechowywania i przeszukiwania.

Tak naprawdę wartość rejestrowania zależy od kontekstu. Podczas debugowania konkretnego problemu w określonym komponencie szczegółowy i obszerny zapis w logach może być kluczowy dla zidentyfikowania problemu. Natomiast podczas standardowego działania, gdy wszystko funkcjonuje poprawnie, większość logów to po prostu zbędny szum, z wyjątkiem tych, które wyzwalają alerty z powodu błędów systemu.

Dlatego tylko najprostsze systemy rozproszone rejestrują zdarzenia za pomocą podstawowego wyświetlania. Prawie każdy system korzysta z jakiejś biblioteki zapisu w logach. Biblioteki te dostarczają ważne funkcje, które znacznie ułatwiają radzenie sobie z dużą ilością logów. Najprostszą funkcjonalnością oferowaną przez takie biblioteki są podstawowe znaczniki czasowe. Często próbujemy zrozumieć, co się wydarzyło i w jakiej kolejności, a bez wspólnego pojęcia czasu w każdym wierszu dziennika trudno jest określić na przykład czas, jaki upłynął między poszczególnymi logami. Oprócz znaczników czasowych większość bibliotek rejestrowania umożliwia dodawanie innego typu informacji do logów. W każdym systemie wielowątkowym wiele żądań może być przetwarzanych równolegle, co oznacza, że może być generowanych wiele identycznych logów. Aby rozróżnić te logi i zrozumieć, jak odnoszą się do określonego żądania, można dodać identyfikator wątku, identyfikator użytkownika bądź identyfikator żądania, które pomagają odróżnić identyczne logi na tym samym serwerze i przefiltrować je do kontekstu konkretnego żądania. Ostatnią, ale często najcenniejszą usługą, jaką oferują biblioteki rejestrowania, jest zapis na różnych

poziomach w logach. Takie rejestrowanie pozwala wyrazić rodzaj zapisywanych logów. Liczba poziomów często różni się w zależności od biblioteki, ale większość z nich oferuje następujące poziomy:

Debugowanie (Debug)
: Szczegółowe logi, zazwyczaj przydatne tylko podczas dokładnego debugowania.

Informacja (Info)
: Często użyteczne wiadomości związane z działaniem usługi, ale niemówiące o niczym wyjątkowym.

Ostrzeżenie (Warning)
: Coś niepokojącego, ale niekoniecznie śmiertelnego dla całej aplikacji.

Błąd (Error)
: Wydarzyło się coś bardzo złego i ktoś powinien jak najszybciej to sprawdzić.

Fatalny błąd (Fatal)
: Wystąpiło coś nieodwracalnego i aplikacja zakończy działanie po zapisaniu tego logu.

Zapisy w logach z określeniem poziomu pozwalają odfiltrować tylko określone typy wiadomości i pozostawić szczegółowe rejestrowanie w kodzie, gdy jest to potrzebne, bez zapełniania strumieni logów wieloma zaśmiecającymi wiadomościami. Zazwyczaj poziom zapisu w logach dla aplikacji jest ustawiany na *Info* lub *Warning*. Wiadomości na poziomie *Debug* można włączyć w każdej chwili w celu diagnozowania określonych problemów.

Oprócz żądania cennym kontekstem dla biblioteki rejestrowania jest miejsce, w którym został wywołany zapis w dzienniku. Zwykle systemy rejestrowania rozumieją, która klasa lub który komponent zawiera log. Umożliwia to **warunkowe rejestrowanie**, w którym dynamicznie włączasz zapis szczegółowych informacji w celu debugowania dla określonych klas lub innych komponentów na działających serwerach. Rejestrowanie warunkowe pozwala reagować na zgłoszone incydenty lub problemy widziane przez danego użytkownika bez konieczności przechowywania i bez opóźnień związanych ze szczegółowym zapisem dla każdego żądania. Należy jednak zachować ostrożność przy rejestrowaniu warunkowym, ponieważ może ono znacznie zwiększać opóźnienia żądań użytkownika końcowego. Nie ma nic gorszego niż spotęgowanie problemu podczas analizy awarii.

Oczywiście aby zapisywanie w logach było użyteczne, musisz znać cel. Ostatecznie rejestrowanie zdarzeń dostarcza wgląd w działanie usługi lub aplikacji. Programiści żartobliwie mówią, że aby wiedzieć, co rejestrować, trzeba przypomnieć sobie, jakich informacji brakowało, gdy początkowo pracowaliśmy nad rozwiązaniem problemu i utyskiwaliśmy: „Szkoda, że w logach nie jest zapisane…". Oczywiście najbardziej przydatne informacje to zdarzenia nietypowe i błędy. Każdy błąd lub wyjątek powinien być zarejestrowany, chyba że jesteś całkowicie pewny obsługującego je kodu (choć i wtedy lepiej jednak je rejestrować). Poza błędami warto rejestrować dziwne zdarzenia. Na przykład niektóre żądania są po prostu wolne. Możesz analizować wskaźniki (zobacz punkt „Wskaźniki"), które pokazują, że 99. percentyl żądań zajmuje absurdalnie dużo czasu, ale bez logów wskazujących, które części obsługi żądań są wyjątkowo wolne, trudno jest podjąć działania. Z drugiej strony samo rejestrowanie szczegółów („wywołano funkcję foo") rzadko bywa przydatne i na pewno zaśmieci logi. Decydując, co zapisywać w logach, wyobraź sobie najtrudniejszy problem, jakiemu będziesz musiał stawić czoła w danej części kodu, i zapisz to, co byłoby Ci potrzebne, aby go skutecznie rozwiązać.

Wskaźniki

W przeciwieństwie do zapisywania w logach, które koncentruje się na pojedynczych wykonaniach, wskaźniki monitorujące są zaprojektowane do zbierania statystyk w czasie, które obrazują ogólne działanie usług. W tym punkcie skupimy się na aplikacji monitorującej Prometheus (*https://prometheus.io*), która stała się standardem w branży. Inne rozwiązania monitorujące oferują podobne możliwości.

Dane monitorujące są zazwyczaj rejestrowane w czasie, gdyż z tej perspektywy można zobaczyć zarówno bieżący stan aplikacji, jak i jego zmienność w czasie.

Podczas monitorowania aplikacji istnieją trzy podstawowe typy danych, które warto rejestrować:

- histogramy,
- liczniki,
- wartości.

Histogramy reprezentują rozkład wartości w określonym przedziale czasowym. Umożliwiają one zrozumienie zarówno średnich, jak i ekstremalnych wrażeń użytkowników. Na przykład histogram opóźnień może pokazać średnią

najlepszych 10% i najgorszych 10% opóźnień, jakich doświadczyli użytkownicy. Takie informacje są przydatne, aby uzyskać pierwsze ogólne wyobrażenie o wydajności aplikacji.

Liczniki to wartości rosnące monotonicznie, które śledzą inne rosnące wartości. Jednym z najczęstszych przykładów licznika jest całkowita liczba żądań obsłużonych przez usługę. Ta liczba może wyłącznie rosnąć w czasie, ponieważ jest coraz więcej żądań. Liczniki często są przekształcane we **współczynniki**, które są pierwszą pochodną licznika i reprezentują współczynnik zmian tej wartości w czasie. Na przykład gdy mamy licznik śledzący całkowitą liczbę żądań, taki współczynnik może pokazać liczbę żądań na sekundę. Liczniki są z natury liczbami całkowitymi.

Wartości natomiast to liczby, które mogą przyjmować dowolne wartości i zarówno rosnąć, jak i maleć w czasie. Typowe przykłady wartości to użycie CPU i pamięci dla usługi. Oczywiście wielkość użycia CPU i pamięci może się zmieniać w czasie — rosnąć lub maleć w zależności od działania usługi. Wartości są najbardziej użyteczne, gdy są obserwowane w czasie, ponieważ mogą wskazywać ogólne zachowanie systemu. Stale rosnące użycie pamięci może wskazywać na wyciek pamięci. Okresowo skokowe użycie CPU może wskazywać na jakiś proces w tle, który jest niepotrzebnie kosztowny.

W tym momencie powinno być jasne, że główną wartością danych monitorujących jest perspektywa czasowa. Rzeczywiście większość monitorowania określa się jako **szereg czasowy** reprezentujący dokładnie te wartości w czasie. Dla tych szeregów czasowych zostały opracowane specjalne bazy danych, aby wydajnie przechowywać dane i wysyłać zapytania. System Prometheus ma prostą bazę danych dla szeregów czasowych, ale przeważnie integruje się go z inną bazą danych przeznaczoną dla szeregów czasowych. Często dobrym rozwiązaniem jest zintegrowanie Prometheusa z opartym na chmurze monitorowaniem jako usługą. Obecnie większość usług monitorujących opartych na chmurze, na przykład Azure Monitor, wspiera importowanie wskaźników z systemu Prometheus.

Jeśli chodzi o pozyskiwanie wskaźników, Prometheus jest systemem opartym na pobieraniu (ang. *pull-based*) wskaźników. Co jakiś czas Prometheus wysyła żądania do Twojej aplikacji w celu uzyskania aktualnego stanu wszystkich wskaźników. Te wskaźniki są przechowywane w bazie danych szeregów czasowych, do której możesz wysyłać zapytania. Monitorowanie oparte na pobieraniu działa dobrze dla aplikacji, które działają zawsze, ale jak rejestrować wskaźniki dla zadań wsadowych i innych procesów przejściowych, które mogą nie działać

wystarczająco długo, aby można było z nich pozyskać dane? Dla takich zadań musisz zaimplementować wskaźniki oparte na wypychaniu (ang. *push-based*). Na szczęście Prometheus ma projekt siostrzany, Prometheus Pushgateway, który implementuje właśnie ten wzorzec. Prometheus Pushgateway uruchamia specjalny serwer, z którego Prometheus może pozyskiwać dane. Twoja aplikacja może wysyłać, czyli „wypychać" wskaźniki do tej bramy sieciowej (ang. *gateway*), aby można je było zarejestrować, nawet jeśli aplikacja już nie działa. Połączenie monitorowania opartego na pobieraniu dla długo działających aplikacji ze wskaźnikami opartymi na wypychaniu dla procesów przejściowych pozwala uzyskać pełny obraz działania aplikacji.

Podstawowe monitorowanie żądań

Jednym z najpopularniejszych typów monitorowania jest monitorowanie liczby żądań obsłużonych przez aplikację. Jest to podstawowa statystyka, która pokazuje, ile osób (lub innych usług) korzysta z Twojej usługi na przestrzeni czasu. Liczba żądań jest przykładem licznika. Nie może maleć, ponieważ nie ma czegoś takiego jak ujemne żądanie.

Jednak sumaryczna liczba żądań nie jest aż tak interesująca. Ciekawsza jest zmiana całkowitej liczby żądań. Wielkość zmiany to wskaźnik wtórny, zdefiniowany jako różnica między liczbą żądań w czasie t a liczbą żądań w czasie $t + 1$. I tak żądania na minutę to całkowita liczba żądań w danej chwili minus całkowita liczba żądań sprzed 1 minuty. Na potrzeby wykresu ta wielkość jest obliczana dla każdej poprzedniej minuty w jednominutowych odstępach. Oczywiście 1 minuta to tylko przykładowy przedział czasowy — takie same obliczenia można wykonać dla dowolnego interwału, od milisekundy do dnia (lub dłuższego).

Obliczenie liczby żądań dla milisekundy stanowi pewne wyzwanie. Jeśli serwer obsługuje mniej niż 1000 żądań na sekundę, obliczanie liczby żądań z dokładnością do milisekundy da bardzo niestabilny wskaźnik, który będzie oscylował między 0 a 1 (może więcej, jeśli nastąpi krótki wzrost żądań). Takie współczynniki są zmienne i w dużej mierze zależą od czynników zewnętrznych. Dlatego często bardziej użyteczne jest korzystanie ze średniej liczby żądań na interwał czasowy niż jedynie z wartości sumarycznej. Uśrednianie tego współczynnika w czasie wygładza szczyty i doliny jego wykresu, dając bardziej realistyczny obraz tego, jak wykorzystywana jest usługa.

Całkowita liczba żądań to dobra informacja na początek, ale aby naprawdę zrozumieć działanie systemu, warto przyjrzeć się bardziej szczegółowym danym każdego żądania. Jedną z najczęstszych cech żądania jest kod odpowiedzi HTTP, który został zwrócony użytkownikom. Jeśli nie znasz jeszcze kodów odpowiedzi HTTP, są one częścią protokołu HTTP, który wskazuje, jak serwer zareagował na żądanie. Do najczęstszych należą 200 (OK), 404 (nie znaleziono) i 500 (błąd wewnętrzny serwera), ale istnieje wiele innych kodów odpowiedzi dla różnych sytuacji. W przypadku wskaźników systemu Prometheus wartości mogą mieć etykiety, które dodają szczegóły do wskaźnika. Korzystając z tej funkcji, możemy traktować kod odpowiedzi jako etykietę związaną z liczbą żądań. Z użyciem tej etykiety możemy zapytać o całkowitą liczbę żądań, które zostały obsłużone pomyślnie (`code==200`), lub o całkowitą liczbę żądań, które zakończyły się błędami (`code==500`). Ponadto możemy obliczyć miarę każdej z tych kategorii żądań, używając wcześniej opisanych metod. Podział całkowitej liczby żądań według kodu odpowiedzi daje nam znacznie lepszy wgląd w działanie aplikacji. Możemy również grupować kody błędów i zobaczyć, ile żądań zakończyło się błędem (kody zaczynające się od 5*xx*), a ile sukcesem (kody zaczynające się od 2*xx*).

Oprócz kodu odpowiedzi dla żądania serwera kolejną najważniejszą cechą żądania jest jego opóźnienie, czyli czas, w jakim dane żądanie zostało wykonane. Kuszące może być traktowanie opóźnienia jako etykiety liczby żądań, ale to nie zadziała. Etykiety muszą być ograniczonym zbiorem dyskretnych wartości, natomiast opóźnienie może przyjmować wiele różnych wartości. Opóźnienie żądania stanowi osobny wskaźnik i jest zazwyczaj modelowane jako histogram. Dzięki histogramowi dla opóźnienia żądań można zrozumieć średnie wrażenia użytkownika, a także wrażenia użytkowników, którzy mieli najwolniejsze żądania. Podobnie jak w przypadku liczby żądań opóźnienie żądań można też podzielić ze względu na kod odpowiedzi. Dzięki temu nie tylko można zobaczyć średnie wrażenia dla dowolnego żądania, ale także obliczyć średnie opóźnienie błędu lub żądania nieznalezionego czy innego kodu odpowiedzi HTTP.

Zaawansowane monitorowanie żądań

Chociaż całkowita liczba żądań, ich szybkość i opóźnienie są najczęstszymi i najbardziej przydatnymi wskaźnikami do monitorowania żądań do serwera lub usługi, istnieją inne, bardziej wyspecjalizowane wskaźniki, które mogą być niezwykle pomocne w dogłębnym zrozumieniu wydajności usługi. Dobrym przykładem są rozmiary żądań i odpowiedzi podane w bajtach. Rozmiar żądania

jest przydatny, ponieważ można go powiązać z wolnymi żądaniami (większe żądania mogą wymagać więcej czasu na analizę i przetworzenie) oraz użyć do obserwowania tendencji w czasie, takich jak rosnące rozmiary żądań lub odpowiedzi. Podobnie jak opóźnienie rozmiar żądania i odpowiedzi to przykłady wskaźników przedstawianych w postaci histogramu.

Niekiedy użyteczne wskaźniki są rejestrowane z dodaniem szczegółów na temat sposobu, w jaki obliczasz i raportujesz wskaźniki w swojej aplikacji. Na przykład chcemy wiedzieć, ile czasu żądanie spędziło w kolejce w porównaniu do czasu spędzonego na samym przetwarzaniu żądania. Tego typu informacje są szczególnie przydatne, gdy szukasz przyczyn powolnego przetwarzania żądań przez usługę. Widząc jedynie, iż opóźnienie jest wysokie, możesz stwierdzić, że istnieje problem i że użytkownicy mogą być niezadowoleni, ale wskaźniki dotyczące czasu w kolejce i czasu przetwarzania pomogą Ci zrozumieć, czy trzeba skalować usługę (czas w kolejce jest zbyt długi), czy zoptymalizować kod (czas przetwarzania jest zbyt długi). Aby uzyskać te wskaźniki, musisz dodać nowe wskaźniki do kodu. W tym przypadku dodałbyś dwa nowe wskaźniki: jeden mierzący czas od otrzymania żądania do rozpoczęcia przetwarzania, a drugi mierzący czas od rozpoczęcia przetwarzania do zakończenia żądania. W ten sposób system będzie miał trzy wskaźniki mierzące opóźnienia: całkowite opóźnienie, opóźnienie w kolejce i opóźnienie przetwarzania. Wskaźniki można dodać stosunkowo łatwo, a dodatkowa wiedza, jakiej dostarczają, bywa bezcenna przy debugowaniu usługi. Oczywiście, jak we wszystkim, można łatwo przesadzić. Jeśli monitorujesz opóźnienie każdej linii kodu (lub nawet każdej funkcji), raczej nie jest to dobry pomysł.

Inny zaawansowany wskaźnik może dotyczyć żądań z podziałem według geografii lub klienta. Takie monitorowanie może dostarczyć cennych informacji biznesowych, które pomogą w tworzeniu lepszych produktów lub zidentyfikowaniu nadużyć pochodzących od konkretnych użytkowników bądź lokalizacji. Jeśli zdecydujesz się na analizę pod kątem geografii (na przykład według kraju), możesz dodać odpowiednią etykietę (na przykład „geografia") do wskaźnika liczby żądań. Jednak częściej tego typu dane muszą być uzyskiwane poprzez wysyłanie zapytań do bazy danych z logami, zamiast bezpośredniego patrzenia na wskaźniki. Na przykład w każdym dużym systemie jest zbyt wielu klientów, aby używać identyfikatora klienta jako etykiety dla wskaźnika liczby żądań. Dlatego system zapisuje identyfikator klienta za każdym razem, gdy żądanie jest przetwarzane. Ten przykład ilustruje wzajemne oddziaływanie między zapisywaniem w logach a monitorowaniem. Monitorowanie i wskaźniki mogą dostarczać danych zbiorczych, które można szybko uzyskać. Natomiast rejestrowanie i

przeszukiwanie logów to technika, która może jest wolniejsza, ale zapewnia dostęp do bardziej szczegółowych danych. Ponieważ analiza biznesowa w większości przypadków nie wymaga danych na już i może na nie poczekać nieco dłużej, pozyskiwanie informacji o użytkowaniu aplikacji przez klientów z logów może lepiej spełniać wymagania.

Alarmowanie

Dotychczas omówiłem różne sposoby rejestrowania informacji z logów i wskaźników aplikacji. Te dane mogą być kluczowe dla zrozumienia, dlaczego system nie działa poprawnie w czasie awarii lub innego incydentu. Ale jak można od razu się dowiedzieć, że coś idzie źle?

Oczywiście pierwszym sposobem, w jaki dowiadujemy się, że w systemie wystąpił problem, jest zgłoszenie od klienta lub użytkownika. Jednakże czekanie na **incydenty zgłaszane przez klientów** (ang. *customer reported incidents*, CRI) to zła praktyka zarówno dla użytkownika, jak i dla osoby otrzymującej powiadomienie. Nic nie podkopuje zaufania klienta bardziej niż informacja, że to on pierwszy zauważył problem. W dzisiejszych czasach każdy użytkownik oczekuje, że zauważymy (i naprawimy) problemy w usługach, zanim użytkownik dostrzeże, iż coś jest nie tak.

Dlatego najlepszym sposobem na wykrycie problemu jest **wysyłanie alertów** przez system. Alarmowanie to praktyka polegająca na ustaleniu warunków, w których zespół odpowiedzialny za daną usługę jest powiadamiany, zwykle poprzez wysłanie alertu, o problemie z systemem.

Podstawowe alarmowanie

Najprostsza forma alarmowania opiera się na sprawdzaniu wskaźników i ustalonych progów. Na przykład możesz ustalić, że system ma wysyłać alerty, jeśli opóźnienie na poziomie 90. percentyla w systemie przekroczy pół sekundy. Możesz także ustawić alert, jeśli liczba błędów HTTP 500 zgłaszanych przez system przekroczy 1% wszystkich żądań. Mając tak ustalone wartości progowe, łatwo zauważyć, że któraś z nich została przekroczona i należy poinformować o tym inżynierów. Ale jak określić właściwe progi alertów dla systemu?

Najprostsze podejście do tworzenia alertów to określenie docelowego poziomu usługi (ang. *service level objective*, SLO) dla aplikacji. Jeśli nie ustawisz alertu, nie zauważysz problemu. Progi alertów definiują różnicę między normalnym a nietypowym działaniem. Każda aplikacja jest inna, dlatego warto zapisać, jakie są cele związane z „normalnymi" wrażeniami użytkownika Twojej usługi, i ustawić alerty, które zadziałają, jeśli te cele nie są osiągane. Kolejnym czynnikiem do rozważenia jest doświadczenie inżynierów pełniących dyżury. Ciągłe uruchamianie się alertów prowadzi do wypalenia zawodowego i rezygnacji. Podobnie częste uruchamianie się alertów będących fałszywymi alarmami spowoduje, że inżynierowie przestaną na nie reagować. Dlatego musisz zrównoważyć cele usługi z potrzebami dyżurnych inżynierów, aby określić właściwe reguły alarmowania. Alerty są tworzone przez cały okres życia usługi. W miarę dodawania nowych funkcjonalności, a co za tym idzie, szerszego monitorowania, należy także dodawać więcej alertów. Z czasem Twoje doświadczenie i wiedza o danym systemie się zwiększają, co możesz wykorzystać przy dostosowywaniu alertów, aby były bardziej restrykcyjne albo luźniejsze, w zależności od potrzeb użytkowników lub dyżurujących inżynierów.

Alarmowanie o anomaliach

Podstawowe alarmowanie w wielu przypadkach się sprawdza, ale czasami okazuje się niewystarczające. Dobrym przykładem jest błąd, który powoduje całkowitą awarię dla małej (1%) grupy użytkowników. Z perspektywy statycznego alarmowania system działa w określonych granicach, ale dla użytkowników z tej grupy usługa jest całkowicie niedostępna. Statyczne alarmowanie zakłada, że dla wszystkich użytkowników i wywołań usług wszystkie wskaźniki są takie same. Jednak nie zawsze tak jest. Wykrywanie anomalii to zadanie, które możemy zlecić sztucznej inteligencji. Na przykład klient *A* ma całkowitą awarię, podczas gdy dla klienta *B* aplikacja działa bez zarzutu. Chociaż alarmowanie na podstawie anomalii jest zaawansowanym tematem, często jest to jedyny sposób na znalezienie równowagi między alarmowaniem w przypadku problemu a generowaniem wielu fałszywych alertów. W miarę jak system się rozrasta i zwiększa się jego złożoność, alarmowanie oparte na wykrywaniu anomalii staje się wymogiem.

Monitorowanie i zapisywanie w logach zapewnia wgląd w aplikację. Alerty informują, kiedy powinieneś zwrócić uwagę na swój produkt. Ważne jest, aby pamiętać, że nie każdy wskaźnik wymaga alertu, niektóre bowiem są użyteczne jedynie do debugowania i wizualizacji. Jednak wydajność aplikacji będzie definiowana przez jakość alertów. Na alarmowanie poświęć tyle czasu i wysiłku co na projektowanie funkcji skierowanych do klienta.

Śledzenie

Jak widzieliśmy w poprzednich rozdziałach, większość nowoczesnych aplikacji obejmuje żądania, które rozciągają się na wiele maszyn i procesów. Z wykorzystaniem opisanych technik możesz monitorować wszystkie te komponenty pojedynczo. Niestety jednak podejście polegające na analizie usługa po usłudze pokazuje żądania przepływające przez system w tym samym czasie, ale nie umożliwia dobrego śledzenia ścieżki pojedynczego żądania przez wszystkie mikrousługi w systemie. Śledzenie żądań obejmuje wszystkie wskaźniki usług i daje widok pełnej ścieżki.

Do śledzenia najprościej jest użyć unikatowego identyfikatora korelacji dla każdego żądania wchodzącego do systemu. Ten identyfikator to liczba, która może zostać użyta do grupowania logów, wskaźników i innych informacji w aplikacji. Identyfikator korelacji może być dowolny, pod warunkiem że unikalnie identyfikuje każde żądanie. Najprostszym identyfikatorem korelacji byłaby duża losowa liczba, ale lepszym rozwiązaniem jest zastosowanie funkcji skrótu do unikatowych cech oryginalnego żądania, takich jak jego ścieżka żądania i adres IP źródła.

Gdy masz identyfikator korelacji, ilekroć zapisujesz log lub wskaźnik, zapisujesz także identyfikator korelacji. Na przykład w systemie zapisującym logi, oprócz sygnatury czasowej i poziomu rejestru, dodawalibyśmy także identyfikator korelacji żądania. Aby umożliwić zapisywanie w logach tego identyfikatora we wszystkich komponentach systemu, musisz przekazywać go wraz z wywołaniami API, które wykonujesz w całym systemie.

Jeżeli wysyłasz żądania API przy użyciu protokołu HTTP, identyfikator korelacji możesz osadzić w nagłówku szczególnym dla aplikacji, na przykład `x-myapps-correlation-id`, i zmodyfikować kod, aby odczytywał identyfikator z tego nagłówka i wypełniał nim obiekt kontekstu żądania. Gdy masz identyfikatory korelacji w logach i wskaźnikach, możesz zgrupować te informacje, aby podejrzeć ścieżkę pojedynczego żądania przez cały system.

Jeśli wydaje Ci się, że to dużo pracy, masz rację. Na szczęście OpenTelemetry (*https://opentelemetry.io*) wykona dużą jej część za Ciebie. OpenTelemetry ma SDK (ang. *software development kit* — zestaw narzędzi dla programistów) dla wszystkich popularnych języków, dzięki czemu łatwiej jest zintegrować śledzenie żądań z aplikacją. Projekt ten jest niezależny od dostawcy, co pozwala na integrację z różnymi backendami do monitorowania wskaźników i rejestrowania w logach. Ponadto zintegrowane z OpenTelemetry liczne narzędzia do wizualizacji

i debugowania dają bogate możliwości introspekcji. W niemal wszystkich przypadkach integracja z otwartoźródłowym projektem realizowanym przez społeczność, takim jak OpenTelemetry, w celu śledzenia żądań jest lepszym pomysłem niż budowanie własnego rozwiązania.

Agregowanie informacji

Kiedy zaczynasz monitorowanie, odkrywasz, że proces ten może generować ogromną ilość informacji. Wydajne utrzymywanie niezbędnych danych oraz możliwość szybkiego przeszukiwania ich pod kątem odpowiednich informacji dla bieżącego problemu stają się wyzwaniem. Agregacja pozwala grupować informacje w taki sposób, aby można było je łatwiej zrozumieć i efektywniej przechowywać.

Pierwsza forma agregacji, od której większość osób zaczyna, to przeszukiwanie logów w wielu procesach. W części dotyczącej śledzenia wyjaśniłem, jak OpenTelemetry zapewnia widok zorientowany na żądania, ale to nie jedyna agregacja, którą możesz wykonać. Jeśli masz komponent, który został zreplikowany podczas skalowania, możesz chcieć wyszukiwać przypadki danego błędu lub awarii we wszystkich replikach.

Na szczęście istnieje wiele narzędzi, które ułatwiają grupowanie logów w wielu kontenerach Kubernetesa. Jednym z najprostszych w użyciu jest narzędzie `ktail`, które rozszerza tradycyjny interfejs `kubectl` (*https://oreil.ly/ktail*), aby łączyć logi z wielu podów. Bardziej zaawansowane narzędzia wprowadzają logi do indeksu wyszukiwania, takiego jak Elasticsearch (*https://oreil.ly/elasticsearch*), co pozwala wysyłać dowolne zapytania w logach, podobnie jak w wyszukiwarce internetowej.

Ustawienie i utrzymanie indeksu logów jest skomplikowane i podatne na błędy, ale na szczęście obecnie wszystkie chmury publiczne i wiele startupów oferuje usługę przesyłania i wyszukiwania logów. Podobne usługi są dostępne dla wskaźników szeregów czasowych. Niezależnie od tego, czy budujesz w chmurze publicznej, czy na własnej infrastrukturze, korzystanie z usługi do czegoś tak istotnego jak logi jest w większości przypadków właściwym wyborem.

Dla logów i wskaźników można używać systemów zaprojektowanych do efektywnego ich przechowywania i przeszukiwania, czasami jednak konieczne jest ograniczenie ilości gromadzonych informacji. Zabieg ten nazywa się downsamplingiem (zmniejszeniem częstotliwości próbkowania) lub agregacją. Najprostszą formą

tej techniki dla wskaźników jest rzecz jasna rzadsze ich zapisywanie. Na przykład jeśli zapisujesz wskaźnik co sekundę, po tygodniu możesz zredukować częstotliwość do minuty. Nawet taka prosta zmiana zmniejsza 60-krotnie miejsce potrzebne do przechowywania danych. Oczywiście takie zmniejszenie częstotliwości próbkowania może prowadzić do utraty informacji. Inną opcją jest użycie wartości średniej z owych 60 sekund zamiast dowolnej wartości chwilowej. Taki prosty downsampling można zastosować do logów. Najprostszym podejściem jest zachowanie tylko określonej klasy logów, takiej jak błędy, i pominięcie mniej ważnych informacji. Możesz także zachować tylko część logów, choć w ten sposób można utracić wiele informacji. Choć to wymaga większego nakładu pracy, możesz opracować agregacje, które zachowują informacje istotne dla Twojej aplikacji, jednocześnie kompresując dane potrzebne do ich przechowywania.

Zmniejszanie częstotliwości zapisywania informacji jest często dobrym rozwiązaniem. Jednak jeśli chcesz obniżyć koszty, zachowując zarazem pełną wierność danych, na przykład ze względu na zgodność z przepisami lub bezpieczeństwo, możesz zamiast tego zmienić dostępność danych. Przesyłanie logów do opartej na chmurze „zamrażarki" może znacząco obniżyć koszty przechowywania danych, jednak spowoduje wolniejszy i trudniejszy dostęp do informacji z tych logów. Może to być dobre rozwiązanie, gdy dane muszą być przechowywane przez długi czas, ale są rzadko używane. Niektóre rozwiązania typu rejestrowanie w logach jako usługa mogą automatycznie implementować taki rodzaj segregacji danych.

Podsumowanie

Chociaż monitorowanie samo w sobie nie przyczynia się do prawidłowego działania systemu, to gdy system działa nieprawidłowo, jest ono kluczowym elementem, który pozwala zidentyfikować, co uległo awarii i jak to naprawić. Umiejętność odpowiedniego monitorowania i obserwowania systemów jest niezbędna, aby zapewnić szybkie przywrócenie poprawnego działania w obliczu problemu.


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion

Brendan Burns prezentuje praktyczne wzorce i paradygmaty projektowe, które są niezbędne do budowy nowoczesnych aplikacji natywnych dla chmury!

— **Lalithkumar Prakashchand,** programista w Meta Platforms

Upowszechnienie kontenerów i narzędzi do ich orkiestracji zrewolucjonizowało rozwój systemów rozproszonych. Dziś projektanci mają do dyspozycji gotowe obiekty, interfejsy, a także bogate zestawy wzorców. Dzięki temu mogą budować komponenty nadające się do wielokrotnego użytku i łatwo skalować całe systemy.

To drugie, zaktualizowane wydanie popularnego podręcznika przedstawia bogatą kolekcję wzorców, dzięki którym tworzenie niezawodnych aplikacji rozproszonych staje się prostsze i bardziej efektywne. To niezbędny przewodnik dla każdego architekta i inżyniera oprogramowania, który chce budować skalowalne i odporne na awarie aplikacje natywne dla chmury.

W książce:

- koncepcje i podstawowe pojęcia dotyczące systemów rozproszonych
- wzorce jednowęzłowe
- wzorce serwowania usług i programowanie zdarzeniowe
- wzorce wsadowego przetwarzania danych
- infrastruktura sztucznej inteligencji
- typowe błędy projektowe, monitorowanie aplikacji i reakcje na typowe awarie

Lektura obowiązkowa dla inżynierów odpowiedzialnych za niezawodność witryn internetowych, a także programistów, którzy chcą w pełni wykorzystać możliwości systemu Kubernetes.

— **Swapnil Shevate,** inżynier i pasjonat niezawodnych witryn internetowych

Dr Brendan Burns jest wiceprezesem Microsoftu odpowiedzialnym za usługi: Azure, Azure Arc, Kubernetes w Azure, Linux w Azure i PowerShell. Wcześniej pracował w Google Cloud, gdzie współtworzył projekt *open source* Kubernetes. Współautor książek *Kubernetes. Tworzenie niezawodnych systemów rozproszonych* i *Najlepsze praktyki w Kubernetes. Jak budować udane aplikacje.*

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 978-83-289-3147-3

Cena: 79,00 zł