O'REILLY®

Wydanie II

Prometheus w pełnej gotowości

Jak monitorować pracę infrastruktury i wydajność działania aplikacji

Julien Pivotto
Brian Brazil

Helion

Tytuł oryginału: Prometheus: Up & Running:
Infrastructure and Application Performance Monitoring, 2nd Edition

Tłumaczenie: Robert Górczyński

ISBN: 978-83-289-0529-0

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/prome2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
https://ftp.helion.pl/przyklady/prome2.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 Prometheus?

Prometheus to bazujący na wskaźnikach system monitorowania, który jest dostępny jako oprogramowanie typu open source. Oczywiście Prometheus nie jest jedynym rozwiązaniem tego typu, więc być może zastanawiasz się nad tym, co go wyróżnia spośród innych.

Prometheus zajmuje się wykonywaniem tylko jednego zadania i robi to świetnie. Posiada prosty i jednocześnie oferujący potężne możliwości model danych i język wykonywania zapytań, który pozwala na sprawdzanie wydajności działania aplikacji i infrastruktury. Nie wykracza poza obszar wskaźników i nie próbuje rozwiązywać problemów, pozostawiając te działania innym narzędziom, które są lepiej przystosowane do tego celu.

Społeczność i ekosystem oprogramowania Prometheus znacznie się zwiększyły od chwili, gdy na początku w 2012 roku w społeczności SoundCloud pracowała zaledwie garstka programistów. System Prometheus został opracowany w języku Go i pozostaje dostępny na licencji Apache 2.0. Setki osób uczestniczą w pracy nad projektem, który nie jest kontrolowany przez żadną firmę. Zawsze trudno jest ustalić, ile użytkowników korzysta z projektu typu open source, ale w 2022 roku szacowano, że setki tysięcy organizacji używało systemu Prometheus w środowisku produkcyjnym. W 2016 roku projekt Prometheus stał się drugim[1] projektem należącym do fundacji Cloud Native Computing Foundation (CNCF).

Na potrzeby koordynacji własnego kodu istnieją biblioteki klienckie opracowane dla praktycznie wszystkich najpopularniejszych języków programowania i środowisk uruchomieniowych, takich jak Go, Java/JVM, C#/.Net, Python, Ruby, Node.js, Haskell, Erlang i Rust. Wiele popularnych aplikacji, np. Kubernetes, Docker, Envoy i Vault, jest już dostarczanych z bibliotekami klienckimi dla oprogramowania Prometheus. Dla innych firm tworzących oprogramowanie udostępniające wskaźniki w formacie innym niż zgodny z systemem Prometheus dostępne są setki rozwiązań integracyjnych. To są tzw. komponenty eksportujące, które obejmują produkty takie jak HAProxy, MySQL, PostgreSQL, Redis, JMX, SNMP, Consul i Kafka. Przyjaciel Briana dodał nawet obsługę monitorowania serwerów Minecrafta, ponieważ dla niego ważna jest osiągana liczba klatek na sekundę podczas gry.

[1] Pierwszy był produkt Kubernetes.

Format w postaci zwykłego tekstu[2] oznacza, że przekazywanie wskaźników do systemu Prometheus jest bardzo łatwe. Inne systemy monitorowania, zarówno komercyjne, jak i typu open source, zapewniają obsługę tego formatu. Dzięki temu w tych rozwiązaniach można skoncentrować się bardziej na ich podstawowej funkcjonalności, a nie na konieczności poświęcania czasu i wysiłku, aby zapewnić obsługę każdego elementu oprogramowania, który użytkownik będzie chciał monitorować.

Model danych identyfikuje poszczególne szeregi czasowe nie tylko za pomocą nazwy, ale również dzięki nieuporządkowanemu zbiorowi par klucz i wartość określanych mianem etykiet. Język zapytań PromQL pozwala na agregację między tymi etykietami. Dzięki temu można przeprowadzać analizy nie tylko dla poszczególnych procesów, ale także dla centrów danych i usług bądź na podstawie innych zdefiniowanych etykiet. Następnie dane można przedstawić na wykresach dzięki użyciu systemów takich jak Grafana (*https://github.com/grafana/grafana*) i Perses (*https://github.com/perses/perses*).

Ostrzeżenia mogą być zdefiniowane za pomocą dokładnie tego samego języka zapytań PromQL, który jest używany do przygotowywania wykresów. Jeżeli coś można przedstawić na wykresie, można również to wykorzystać w ostrzeżeniach. Dzięki etykietom obsługa techniczna ostrzeżeń staje się łatwiejsza, ponieważ można utworzyć pojedyncze ostrzeżenie obejmujące wszystkie możliwe wartości etykiet. W niektórych systemach monitorowania istnieje możliwość pojedynczego tworzenia ostrzeżeń dla każdego urządzenia i aplikacji. W związku z tym mechanizm wykrywania usług może automatycznie ustalić, które aplikacje i urządzenia powinny być pozyskane ze źródeł takich jak Kubernetes, Consul, Amazon Elastic Compute Cloud (EC2), Azure, Google Compute Engine (GCE) i OpenStack.

Mimo tych wszystkich funkcji i zalet Prometheus jest efektywny i prosty w użyciu. Pojedynczy serwer systemu Prometheus może pobierać miliony próbek na sekundę. Serwer ma postać pojedynczego pliku binarnego ze statycznie dołączonymi bibliotekami oraz pliku konfiguracyjnego. Wszystkie komponenty systemu Prometheus mogą być uruchamiane w kontenerach i unikają wykonywania jakichkolwiek zadań utrudniających działanie narzędziom przeznaczonym do zarządzania konfiguracją. Prometheus został opracowany z myślą o jego zintegrowaniu z istniejącą infrastrukturą i używaniu razem z nią, a nie jako platforma zarządzania.

Skoro już mniej więcej wiesz, czym jest Prometheus, warto zrobić krok wstecz i wyjaśnić, co oznacza pojęcie „monitorowanie”, aby tym samym zapewnić pewien kontekst. Następnie przedstawimy podstawowe koncepcje kryjące się za systemem Prometheus i wyjaśnimy, czym Prometheus nie jest.

Czym jest monitorowanie?

W szkole średniej jeden z nauczycieli Briana powiedział mu, że jeśli zapyta 10 ekonomistów o znaczenie określenia ekonomia, to otrzyma 11 odpowiedzi. W przypadku monitorowania mamy do czynienia z podobnym brakiem zgody co do faktycznego znaczenia tego terminu. Gdy Brian wspomina innym osobom, czym się zajmuje, ludzie są przekonani, że jego praca obejmuje praktycznie wszystko od pilnowania temperatury w fabrykach do śledzenia pracowników, aby dowiedzieć się, kto korzysta z Facebooka w pracy, oraz wykrywać intruzów w sieci firmowej.

[2] Poza formatem zwykłego tekstu została opracowana jeszcze bardziej standaryzowana i nieco odmienna wersja formatu tekstowego systemu Prometheus, która nosi nazwę OpenMetrics.

Prometheus nie został opracowany do wykonywania tego rodzaju zadań[3]. Ten produkt został zbudowany, aby pomóc twórcom oprogramowania i administratorom w nadzorowaniu produkcyjnych systemów komputerowych, np. aplikacji, narzędzi, baz danych i sieci kryjących się za popularnymi witrynami internetowymi.

Czym jest więc monitorowanie w takim kontekście? Warto zawęzić ten rodzaj monitorowania operacyjnego systemu komputerowego do czterech kwestii.

Powiadamianie

Wiedza o tym, kiedy dzieje się coś złego, to zwykle najważniejszy powód, dla którego stosuje się monitorowanie. Oczekuje się, że w przypadku problemu system monitorowania poinformuje o tym człowieka, który będzie mógł odpowiednio zareagować.

Debugowanie

Po poinformowaniu człowiek musi przyjrzeć się problemowi w celu znalezienia jego podstawowej przyczyny i ostatecznie go usunąć.

Trendy

Powiadamianie i debugowanie zwykle odbywają się w czasie wyrażonym w minutach bądź godzinach. Wprawdzie mniej pilna, ale równie użyteczna jest możliwość sprawdzenia, jak systemy są używane na przestrzeni czasu i jak się wówczas zmieniają. Istniejące trendy mogą wpływać na podejmowanie decyzji projektowych, a także planowanie pojemności.

Instalowanie systemu

Gdy do dyspozycji masz jedynie młotek, wtedy wszystko zaczyna przypominać gwoździe. Ostatecznie wszystkie systemy monitorowania danych są tak naprawdę potokami przetwarzania informacji. Czasami wygodniejsze będzie przeznaczenie fragmentu systemu monitorowania do innego celu niż tworzenie oddzielnego rozwiązania. W takim przypadku nie mamy do czynienia z czystym monitorowaniem, ale to jest na tyle często spotykane podejście, że zdecydowaliśmy się na jego omówienie.

W zależności od tego, z kim rozmawiasz i jakie ta osoba ma doświadczenie, tylko część z wymienionych zadań może ona uznać za monitorowanie. To prowadzi do wielu odbywających się w kółko dyskusji dotyczących monitorowania, których efektem jest jedynie frustracja uczestników. Aby pomóc Ci w zrozumieniu perspektywy innych, w kolejnym punkcie zamieszczono nieco historii dotyczącej monitorowania.

Krótka i niepełna historia monitorowania

W ostatnich latach można zauważyć zmianę w monitorowaniu polegającą na przejściu w kierunku narzędzi takich jak Prometheus. Przez długi czas dominującym rozwiązaniem było pewnego rodzaju połączenie Nagios i Graphite lub ich różne warianty.

[3] Monitorowanie temperatury urządzeń i centrów danych jednak nie jest tak rzadko spotykane. Wiemy również o kilku użytkownikach, którzy dla rozrywki wykorzystują system Prometheus do śledzenia pogody.

Wspominając Nagios, mamy na myśli wszelkie oprogramowanie zaliczane do tej samej obszernej rodziny oprogramowania, czyli m.in. Icinga, Zmon i Sensu. Ich działanie polega przede wszystkim na regularnym wykonywaniu skryptów określanych mianem *sprawdzeń*. Jeżeli sprawdzenie kończy się niepowodzeniem i powoduje zwrot niezerowego kodu stanu, wówczas następuje wygenerowanie powiadomienia. Oprogramowanie Nagios zostało początkowo utworzone przez Ethana Galstada w 1996 roku jako aplikacja MS-DOS przeznaczona do wykonywania żądań ping. Najpierw w 1999 roku zostało wydane pod nazwą NetSaint, zaś później w 2002 roku jego nazwa została zmieniona na Nagios.

Historia Graphite wymaga cofnięcia się do 1994 roku, w którym to Tobias Oetiker utworzył skrypt Perla. Następnie w 1995 roku ten skrypt stał się programem Multi Router Traffic Grapher, MRTG 1.0. Zgodnie z nazwą jego podstawowym przeznaczeniem było monitorowanie sieci za pomocą protokołu SNMP (ang. *simple network management protocol*). Dzięki wykonywaniu skryptów ten program mógł również pobierać wskaźniki[4]. W 1997 roku nastąpiły duże zmiany związane z przepisaniem pewnych fragmentów kodu źródłowego w języku C oraz utworzeniem bazy danych Round Robin Database (RRD) przeznaczonej do przechowywania danych wskaźników. W efekcie nastąpiła znaczna poprawa wydajności działania, zaś RRD stała się punktem wyjścia dla innych narzędzi, m.in. Smokeping i Graphite.

Począwszy od 2006 roku, Graphite używa do przechowywania wskaźników produktu o nazwie Whisper, którego budowa jest podobna do bazy danych RRD. Program Graphite sam nie pobiera danych, raczej je otrzymuje przekazywane przez kolekcję narzędzi takich jak collectd i StatsD, utworzonych w odpowiednio 2005 i 2010 roku.

Jak widzisz, generowanie wykresów i powiadamianie były kiedyś oddzielnymi zadaniami, wykonywanymi przez całkowicie odmienne narzędzia. Wprawdzie można było utworzyć skrypt sprawdzenia w celu analizy zapytania w Graphite i generować powiadomienia na podstawie tego skryptu, ale większość sprawdzeń dotyczy nieoczekiwanych stanów, takich jak niedziałający proces.

Inną pozostałością po tamtej epoce jest praktycznie ręczne podejście do administrowania usługami komputerowymi. Usługi były wdrażane w poszczególnych komputerach i starannie obsługiwane przez administratorów systemów. Powiadomienia dotyczące potencjalnych problemów były przekazywane do odpowiednich inżynierów. Gdy popularność zyskała chmura i technologie związane z natywną chmurą, takie jak EC2, Docker i Kubernetes, traktowanie poszczególnych komputerów i usług jak zwierząt domowych i poświęcanie każdej z nich oddzielnej uwagi okazało się niemożliwe. Raczej są one monitorowane i administrowane jako grupa. Tak jak branża informatyczna odeszła od zarządzania ręcznego i przeszła do narzędzi typu Chef i Ansible, a obecnie zaczyna korzystać z technologii takich jak Kubernetes, tak podobne zmiany są zauważalne także w obszarze monitorowania. To oznacza przejście od sprawdzania poszczególnych procesów w pojedynczych komputerach do monitorowania na podstawie stanu usługi jako całości.

[4] Brian ma miłe wspomnienia z początku 2000 roku związane z konfiguracją MRTG i tworzeniem skryptów przeznaczonych do informowania o temperaturze i poziomie użycia sieci moich komputerów domowych.

Powracamy do teraźniejszości. Technologia OpenTelemetry została opracowana na podstawie dwóch innych projektów typu open source, OpenCensus i OpenTracking. OTel[5] to specyfikacja i zbiór komponentów, które mają zaoferować projektom wbudowaną telemetrię. To jest komponent wskaźników zgodny z systemem Prometheus, wzbogacony o komponent pobierający OpenTelemetry[6], odpowiedzialny za pobieranie wskaźników i ich dostarczanie serwerowi systemu Prometheus.

Zwróć uwagę, że nie wspomnieliśmy tutaj o rejestrowaniu danych, śledzeniu i profilowaniu. W przeszłości dzienniki zdarzeń były uznawane za coś, względem czego można ręcznie używać poleceń takich jak `tail`, `grep` i `awk`. Możesz mieć doświadczenie w pracy z narzędziami analizy typu AWStats w celu generowania raportów co godzinę bądź codziennie. W ostatnich latach można zauważyć znacznie częstsze używanie dzienników zdarzeń w trakcie monitorowania np. z zastosowaniem Elasticsearch, Logstash i Kibana (ELK) oraz stosu OpenSearch. Śledzenie i profilowanie, ogólnie rzecz biorąc, odbywa się za pomocą innych stosów oprogramowania: Zipkin i Jeager zostały opracowane na potrzeby śledzenia, podczas gdy Parca i Pyroscope są używane do nieustannego profilowania.

Skoro przedstawiliśmy nieco historii dotyczącej tworzenia wykresów i powiadamiania, warto teraz zobaczyć, jak wskaźniki i dzienniki zdarzeń wpasowują się w tym obszarze. Czy istnieje więcej kategorii monitorowania niż te dwie wymienione?

Kategorie monitorowania

Ostatecznie większość monitorowania dotyczy tego samego: zdarzeń. Te zdarzenia to może być np.:

- otrzymanie żądania HTTP,
- udzielenie odpowiedzi HTTP 400,
- rozpoczęcie wykonywania funkcji,
- osiągnięcie bloku `else` konstrukcji `if`,
- opuszczenie funkcji,
- zalogowanie użytkownika,
- zapisanie danych na dysku,
- odczytanie danych z sieci,
- żądanie od jądra większej ilości pamięci.

Wszystkie zdarzenia posiadają kontekst. Żądanie IP będzie miało adres IP źródłowy i docelowy, żądany adres URL, zdefiniowane ciasteczka oraz użytkownika wykonującego dane żądanie. Odpowiedź HTTP określi, ile czasu trwało udzielenie tej odpowiedzi, jaki jest kod stanu HTTP i jaka jest długość tej odpowiedzi. Zdarzenia obejmujące funkcje posiadają stos wywołań funkcji znajdujących się powyżej oraz wszystko, co wywołało tę część stosu, np. żądanie HTTP.

Posiadanie całego kontekstu dla wszystkich zdarzeń będzie niezwykle pomocne w przypadku debugowania i poznawania wydajności działania systemu w kategoriach zarówno technicznych, jak

[5] OTel to nieformalna nazwa dla OpenTelemetry.

[6] W chwili powstawania książki programiści systemu Prometheus ustalili w trakcie spotkania, że w przyszłości serwer Prometheus będzie zapewniał natywną obsługę protokołu OpenTelemetry. Jednak nie zapadła jeszcze żadna konkretna decyzja, kiedy i jak to będzie się odbywać.

i biznesowych. Jednak przetwarzanie i przechowywanie takiej ilości danych nie jest praktyczne. Dlatego też mamy z grubsza cztery podejścia pozwalające na zmniejszenie ilości danych do poziomu znacznie bardziej praktycznego: profilowanie, śledzenie, rejestrowanie danych i wskaźniki.

Profilowanie

Profilowanie polega na tym, że nie można mieć przez cały czas pełnego kontekstu dla wszystkich zdarzeń, a jedynie pewien fragment kontekstu i to przez ograniczony czas.

Przykładem narzędzia profilowania jest polecenie `tcpdump`. Pozwala ono rejestrować ruch sieciowy na podstawie określonego filtru. Wprawdzie to jest niezwykle ważne narzędzie debugowania, ale tak naprawdę nie będzie działało przez cały czas, ponieważ bardzo szybko doprowadziłoby to do zapełnienia dysku.

Innym przykładem mogą być kompilacje w trybie debugowania plików binarnych pozwalające na śledzenie danych profilowania. Dostarczają one mnóstwa użytecznych informacji, ale spadek wydajności działania podczas zbierania tych wszystkich danych, np. czas wykonywania poszczególnych funkcji, powoduje, że ogólnie rzecz biorąc, nieustanne działanie tego rodzaju narzędzia w środowisku produkcyjnym nie będzie praktyczne.

W jądrze systemu Linux technologia eBPF (ang. *enhanced berkeley packet filters*) pozwala na szczegółowe monitorowanie zdarzeń jądra, począwszy od operacji systemu plików, po działalność sieciową. Ten mechanizm zapewnia dostęp na poziomie, który wcześniej był niedostępny. eBPF oferuje także jeszcze inne zalety, np. przenośność i bezpieczeństwo. Warto zapoznać się z informacjami, które na temat eBPF opublikował Brendan Gregg na stronie *https://www.brendangregg.com/ebpf.html*.

Profilowanie służy głównie do taktycznego debugowania. Jeżeli będzie prowadzone na dłuższą metę, wówczas ilość danych będzie musiała być zmniejszona, aby mieścić się w jednej z innych kategorii monitorowania. W przeciwnym razie trzeba będzie przejść do **nieustannego profilowania**, które umożliwia zbieranie danych w dłuższym czasie.

Nowością w nieustannym profilowaniu jest to, że w celu zmniejszenia ilości danych i zapewnienia względnie niskiego obciążenia zmniejszeniu ulega częstotliwość profilowania. Jednym z zyskujących popularność narzędzi nieustannego profilowania jest bazujące na technologii eBPF oprogramowanie Parca (*https://www.parca.dev/*). Jego agent używa częstotliwości wynoszącej 19 Hz[7]. W efekcie to narzędzie próbuje pobierać statycznie ważne dane na przestrzeni minut zamiast sekund, a jednocześnie dostarcza dane niezbędne do zrozumienia, jak jest używany czas procesora w infrastrukturze, oraz pomaga w poprawieniu efektywności działania aplikacji tam, gdzie to jest konieczne.

Śledzenie

Śledzenie zwykle nie obejmuje wszystkich zdarzeń, a raczej ich pewną część, np. jedno na sto zdarzeń wywoływanych w interesującej nas funkcji. Śledzenie zanotuje funkcje na interesującym Cię stosie wywołań oraz dostarczy informacje o czasie ich wykonywania. Dzięki temu można

[7] Porównaj to z częstotliwością 100 Hz środowiska uruchomieniowego Go lub nawet z częstotliwością 10 000 Hz w Chromium.

sprawdzić, co w programie zabiera najwięcej czasu i które ścieżki wykonywania kodu mają największy wpływ na ogólne opóźnienie działania aplikacji.

Zamiast wykonywać migawki stosu wywołań w interesujących Cię miejscach, niektóre systemy śledzą i rejestrują czas wywoływania wszystkich funkcji znajdujących się na stosie poniżej interesującej Cię funkcji. Na przykład mogą być zebrane dane jednego na sto żądań HTTP, a na ich podstawie można ustalić, ile czasu zajmuje aplikacji komunikacja z backendem takim jak bazy danych i bufory. Dzięki temu można sprawdzić, jak zmienia się czas obsługi w zależności od czynników typu użycie buforowania.

Systemy śledzenia rozproszonego stosują jeszcze inne podejście. Śledzenie odbywa się między różnymi procesami przez dołączanie unikatowych identyfikatorów do żądań przekazywanych między procesami za pomocą zdalnych wywołań procedur (ang. *remote procedure calls*). Dane zebrane z różnych procesów i urządzeń mogą być połączone ze sobą dzięki identyfikatorowi żądania. To jest ważne narzędzie przeznaczone do debugowania rozproszonego architektury mikrousług. Do technologii na tym obszarze zaliczamy OpenZipkin i Jaeger.

W przypadku śledzenia próbkowanie pozwala na zbieranie rozsądnej ilości danych i nie powoduje nadmiernego obciążenia.

Rejestrowanie danych

Rejestrowanie danych pobiera określony zbiór zdarzeń i rejestruje pewien kontekst dla każdego z tych zdarzeń. Na przykład mogą być rejestrowane wszystkie przychodzące żądania HTTP bądź wszystkie wychodzące wywołania do bazy danych. Aby uniknąć użycia zbyt wielu zasobów, zwykle stosuje się ograniczenie do mniej więcej 100 pól dla wpisu dziennika zdarzeń. Po przekroczeniu tej granicy problem może stanowić przepustowość i brak miejsca w systemie pamięci masowej.

Na przykład, jeśli serwer obsługuje 1000 żądań na sekundę, wówczas wpis dziennika zdarzeń zawierający 100 pól, z których każde ma tylko 10 bajtów, będzie wymagał 1 megabajta danych na sekundę. To wcale nie jest mało w przypadku karty sieciowej o przepustowości 100 Mb. Ponadto to oznacza zapisanie dziennie 84 GB danych na potrzeby rejestrowania zdarzeń.

Dużą zaletą rejestrowania danych jest zwykle brak próbkowania zdarzeń, więc nawet w przypadku ograniczonej liczby pól praktyczne będzie określenie, jak powoli wykonywane zapytania wpływają na określonego użytkownika wykorzystującego konkretny punkt końcowy API.

Podobnie jak monitorowanie ma odmienne znaczenia dla różnych osób, tak samo jest w przypadku rejestrowania danych. W zależności od tego, komu zadasz pytanie, otrzymasz odmienne odpowiedzi, co może prowadzić do dezorientacji. Różne rodzaje rejestrowania danych mają odmienne zastosowania oraz wymagania dotyczące trwałości i przechowywania. Jak możesz się przekonać, istnieją cztery ogólne i nieco nakładające się kategorie.

Transakcyjne dzienniki zdarzeń

To są rekordy biznesowe o znaczeniu krytycznym, którym za wszelką cenę trzeba zapewnić bezpieczeństwo, prawdopodobnie na zawsze. Do tej kategorii zaliczamy wszystko, co dotyczy pieniędzy lub jest używane dla dotyczącej użytkownika funkcjonalności o znaczeniu krytycznym.

Dzienniki zdarzeń żądań

Jeżeli śledzisz każde żądanie HTTP lub wywołanie bazy danych, wówczas masz do czynienia z dziennikiem zdarzeń żądań. Mogą być one przetwarzane w celu implementacji funkcjonalności dotyczącej użytkownika bądź też na potrzeby wewnętrznych optymalizacji. Ogólnie rzecz biorąc, nie chcesz ich utracić, ale jeśli coś się z nimi stanie, to jeszcze nie będzie końcem świata.

Dzienniki zdarzeń aplikacji

Nie wszystkie dzienniki zdarzeń dotyczącą żądań, część z nich dotyczy procesów. Typowe są komunikaty początkowe, wykonywane w tle zadania konserwacyjne bądź inne wpisy na poziomie procesu. Te dzienniki zdarzeń są często przeglądane bezpośrednio przez człowieka, więc w przypadku normalnego działania aplikacji należy unikać umieszczania w nich więcej niż tylko kilku wpisów na minutę.

Dzienniki zdarzeń debugowania

Dzienniki zdarzeń debugowania zwykle są bardzo szczegółowe, a tym samym kosztowne do utworzenia i przechowywania. Bardzo często są używane w ściśle określonych sytuacjach debugowania i mają tendencję do profilowania, co wynika z ilości danych. Wymagania dotyczące niezawodności i przechowywania są małe, zaś same dzienniki zdarzeń debugowania mogą nawet nie opuszczać urządzenia, w którym zostały wygenerowane.

Traktowanie poszczególnych rodzajów dzienników zdarzeń w ten sam sposób może być najgorszym z możliwych rozwiązań — otrzymujesz ogromną ilość dzienników zdarzeń debugowania w połączeniu z wyjątkowymi wymaganiami w zakresie niezawodności dotyczącymi dzienników zdarzeń transakcji. Dlatego też wraz ze wzrostem systemu należy zaplanować oddzielenie dzienników zdarzeń debugowania, aby mogły być obsługiwane oddzielnie.

Przykładami systemów rejestrowania danych są stos ELK, OpenSearch, Grafana, Loki i Graylog.

Wskaźniki

Wskaźniki w dużej mierze ignorują kontekst i zamiast tego śledzą agregacje na przestrzeni czasu różnego rodzaju zdarzeń. W celu zapewnienia rozsądnego poziomu użycia zasobów konieczne jest ograniczenie ilości śledzonych danych: 10 000 dla procesu wydaje się rozsądną górną granicą, o której warto pamiętać.

Przykładami rodzajów wskaźników, które możesz mieć, będą liczba otrzymanych żądań HTTP, ilość czasu poświęconego na obsługę żądań, a także liczba aktualnie przetwarzanych żądań. Dzięki wyłączeniu wszelkich informacji o kontekście ilość danych koniecznych do przechowywania i przetwarzania pozostaje na rozsądnym poziomie.

To oczywiście nie oznacza, że kontekst zawsze jest ignorowany. W przypadku żądania HTTP można zdecydować o zachowaniu wskaźników dla poszczególnych ścieżek adresu URL. Pamiętaj o zalecanej granicy 10 000 wskaźników, ponieważ każda oddzielna ścieżka jest uznawana za wskaźnik. Używanie kontekstu w postaci np. adresu e-mail użytkownika nie będzie rozsądnym rozwiązaniem, ponieważ użytkownik może ich mieć nieograniczoną ilość[8].

[8] Adres e-mail zalicza się do danych wrażliwych, co z kolei wiąże się z aspektami zapewnienia zgodności i prywatności, których najlepiej będzie unikać w monitorowaniu.

Wskaźników można używać do śledzenia opóźnienia i ilości danych, które są obsługiwane przez poszczególne podsystemy aplikacji. Dzięki temu łatwiej jest znaleźć przyczynę wszelkich spowolnień działania. Wprawdzie dzienniki zdarzeń nie mogą rejestrować wielu pól, ale gdy już wiadomo, który podsystem odpowiada za dany problem, wówczas dzienniki zdarzeń mogą pomóc w ustaleniu, którym dokładnie żądaniom użytkownika należy się przyjrzeć.

W tym miejscu jest najbardziej widoczny kompromis między dziennikami zdarzeń i wskaźnikami. Wskaźniki pozwalają na zbieranie danych dotyczących zdarzeń we wszystkich procesach, choć ogólnie rzecz biorąc, z nie więcej niż jednym bądź dwoma polami kontekstu o ograniczonej liczności. Z kolei dzienniki zdarzeń pozwalają na zbieranie informacji o wszystkich zdarzeniach jednego typu, ale mogą śledzić sto pól kontekstu o ograniczonej liczności. Trzeba koniecznie zrozumieć kwestie dotyczące liczności i ograniczeń nakładanych na wskaźniki, do tego jeszcze powrócimy w dalszej części książki.

Jako system monitorowania na podstawie wskaźników Prometheus został opracowany do śledzenia ogólnego stanu systemu, sposobu jego działania oraz wydajności działania, a nie poszczególnych zdarzeń. Ujmując rzecz inaczej, można powiedzieć, że dla systemu Prometheus ważne jest, że w ostatniej minucie zostało wykonanych 15 żądań, których obsługa zajęła 4 sekundy, spowodowała 40 odwołań do bazy danych, 17-krotne użycie bufora i dokonanie dwóch zakupów przez klientów. Koszt i ścieżki kodu poszczególnych wywołań to kwestie związane z profilowaniem lub rejestrowaniem danych.

Skoro już wiesz, jak Prometheus mieści się ogólnie w obszarze monitorowania, zapoznaj się z różnymi komponentami tego systemu.

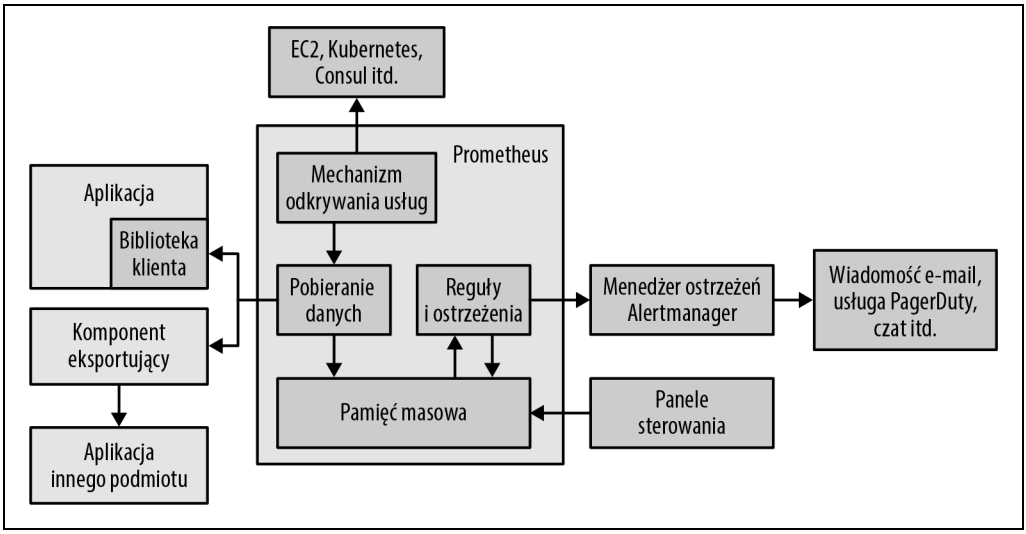
Architektura systemu Prometheus

Na rysunku 1.1 pokazaliśmy ogólną architekturę systemu Prometheus. Komponent odkrywania usług pozwala na znalezienie systemów, z których będą pobierane dane. Celem monitorowania mogą być Twoje własne aplikacje bądź też opracowane przez podmioty zewnętrzne aplikacje, z których dane będą pobierane za pomocą komponentów eksportujących. Zebrane dane są przechowywane w pamięci masowej i można ich używać w panelach sterowania, wykorzystując do tego PromQL, bądź przekazywać ostrzeżenia do menedżera ostrzeżeń Alertmanager, który skonwertuje je na wiadomości dla pagera, wiadomości e-mail bądź innego rodzaju powiadomienia.

Biblioteki klienta

Wskaźniki zwykle nie pojawiają się w magiczny sposób z aplikacji, ktoś musi zapewnić możliwość ich dostarczenia. W tym miejscu do gry wchodzą biblioteki klienta. Zwykle dzięki zaledwie dwóm lub trzem wierszom kodu można zarówno zdefiniować wskaźnik, jak i dodać niezbędną instrumentację w kodzie, nad którym masz kontrolę. To nosi nazwę instrumentacji bezpośredniej.

Biblioteki klienta są dostępne dla wszystkich najważniejszych języków programowania i środowisk uruchomieniowych. Projekt Prometheus dostarcza oficjalne biblioteki klienta dla języków Go, Python, Java/JVM, Ruby i Rust. Dostępnych jest również wiele bibliotek opracowanych przez podmioty zewnętrzne i przeznaczonych dla języków takich jak C#/.Net, Node.js, Haskell i Erlang.



Rysunek 1.1. Architektura systemu Prometheus

Oficjalne kontra nieoficjalne

Nie zniechęcaj się tym, że rozwiązania w zakresie integracji, np. biblioteki klienta, są nieoficjalne bądź zostały opracowane przez podmioty zewnętrzne. Istnieją setki aplikacji i systemów, z którymi być może chcesz przeprowadzać integrację, więc nie ma możliwości, aby zespół tworzący system Prometheus miał czas i doświadczenie wymagane do zapewnienia integracji z tymi wszystkimi aplikacjami i systemami. Dlatego też w ekosystemie oprogramowania Prometheus większość rozwiązań dotyczących integracji została przygotowana i dostarczona przez podmioty zewnętrzne. Aby zachować względną spójność i oczekiwany sposób działania, dostępne są wskazówki wyjaśniające, jak należy opracowywać tego rodzaju rozwiązania w zakresie integracji z aplikacjami i systemami.

Biblioteki klienckie dbają o wszystkie najdrobniejsze szczegóły związane z m.in. zapewnieniem bezpieczeństwa wątków, gromadzeniem danych oraz tworzeniem tekstu systemu Prometheus i/lub udostępnianiem formatu OpenMetrics w odpowiedzi na żądania HTTP. Monitorowanie na podstawie wskaźników nie śledzi poszczególnych zdarzeń, więc poziom użycia pamięci przez bibliotekę klienta nie zwiększa się wraz z liczbą zdarzeń koniecznych do obsłużenia. Zamiast tego poziom użycia pamięci jest powiązany z liczbą posiadanych wskaźników.

Jeżeli jedna z zależności biblioteki aplikacji ma infrastrukturę, zostanie ona automatycznie pobrana. Dlatego też poprzez instrumentację kluczowej biblioteki, np. klienta RPC, można zapewnić instrumentację dla wszystkich aplikacji.

Niektóre wskaźniki, np. poziom użycia procesora i dane statystyczne działania mechanizmu usuwania nieużytków, są zwykle dostarczane standardowo przez biblioteki klientów, w zależności od samej biblioteki i środowiska uruchomieniowego.

Biblioteki klientów nie są ograniczone do dostarczania wskaźników jedynie w formatach tekstowych systemu Prometheus i OpenMetrics. Prometheus to otwarty ekosystem i te same API używane w celu generowania danych w formacie tekstowym mogą być wykorzystane także do tworzenia wskaźników w innych formatach bądź do wprowadzania danych w innych systemach instrumentacji. Podobnie istnieje możliwość pobierania wskaźników z innych systemów instrumentacji i dostarczania bibliotece klienta, jeśli jeszcze wszystko nie zostało do końca skonwertowane na postać instrumentacji systemu Prometheus.

Komponenty eksportujące

Nie wszystkie uruchamiane fragmenty kodu będą kodem, nad którym masz kontrolę lub do którego zachowujesz dostęp. Dlatego też dodawanie bezpośredniej instrumentacji nie zawsze będzie dostępne. Na przykład istnieje małe prawdopodobieństwo, że jądro systemu operacyjnego rozpocznie wkrótce generowanie przez HTTP wskaźników sformatowanych w formacie systemu Prometheus.

Tego rodzaju oprogramowanie często posiada pewne interfejsy, za pomocą których można uzyskać dostęp do wskaźników. To może być format tymczasowy, wdrażany dla aplikacji, z której mają być pobierane wskaźniki.

Komponent eksportujący pobiera żądania od systemu Prometheus, zbiera niezbędne dane z aplikacji, konwertuje je na odpowiedni format, a następnie przekazuje te dane w odpowiedzi przygotowanej dla oprogramowania Prometheus. Komponent eksportujący można traktować jako rodzaj proxy typu jeden do jednego, który konwertuje dane między interfejsem wskaźników aplikacji i formatem systemu Prometheus.

W przeciwieństwie do bezpośredniej instrumentacji stosowanej w przypadku kodu, nad którym zachowujesz kontrolę, komponenty eksportujące używają innego stylu instrumentacji, znanego jako **niestandardowe komponenty pobierające dane** lub *ConstMetrics*[9].

Dobrą wiadomością jest to, że za względu na wielkość społeczności systemu Prometheus potrzebny Ci komponent eksportujący prawdopodobnie już istnieje, a możliwość jego wykorzystania będzie wymagała tylko niewielkiej ilości pracy z Twojej strony. Jeżeli taki komponent nie dostarcza interesującego Cię wskaźnika, zawsze możesz poprosić o jego uzupełnienie i tym samym usprawnienie komponentu dla następnej osoby, która będzie chciała go używać.

Odkrywanie usług

Gdy wszystkie aplikacje zostały przygotowane i działają już komponenty eksportujące, Prometheus musi je znaleźć. Dzięki temu Prometheus będzie wiedział, co ma monitorować, i będzie w stanie zauważyć, że coś, co ma być monitorowane, nie odpowiada. W przypadku środowisk dynamicznych nie można po prostu jednokrotnie dostarczyć listy aplikacji i komponentów eksportujących, ponieważ będzie ona nieaktualna. W tym miejscu do gry wchodzi mechanizm odkrywania usług.

[9] Pojęcie ConstMetrics jest potoczne i wiąże się z funkcją `MustNewConstMetric` biblioteki klienta dla języka programowania Go, która jest używana do wygenerowania wskaźników przez komponenty eksportujące utworzone w kodzie Go.

Prawdopodobnie masz pewną bazę danych urządzeń, aplikacji i wykonywanych przez nie zadań. To może być baza danych Chef, plik zasobów dla Ansible, bazujący na znacznikach w egzemplarzu EC2, w etykietach i adnotacjach w Kubernetes bądź po prostu w dokumentacji wiki.

Prometheus zapewnia integrację z wieloma powszechnie stosowanymi mechanizmami odkrywania usług, takimi jak Kubernetes, EC2 i Consul. Istnieje również ogólne rozwiązanie w zakresie integracji (więcej informacji na ten temat znajdziesz w rozdziale 8.).

Mimo to problem wciąż pozostaje. Nawet jeśli Prometheus posiada listę urządzeń i usług, wcale nie oznacza to, że wiadomo jak one wpasowują się w architekturę. Na przykład możesz używać znacznika EC2 `Name`[10] w celu wskazania aplikacji uruchomionych w urządzeniu, podczas gdy inni mogą używać znacznika o nazwie `app`.

Każda organizacja używa nieco odmiennego rozwiązania. Prometheus pozwala skonfigurować sposób, w jaki metadane z mechanizmu odkrywania usług będą mapowane na systemy przeznaczone do monitorowania i ich etykiety, używając do tego procesu określanego mianem *relabeling*.

Zbieranie danych

Odkrywanie usług i proces relabeling dostarczają listę systemów przeznaczonych do monitorowania. Teraz Prometheus musi pobrać z nich wskaźniki. W tym celu Prometheus wykonuje żądanie HTTP określane mianem *scrape*. Odpowiedź udzielona na to żądanie zostaje przetworzona i zapisana w pamięci masowej. Dodanych zostaje również wiele użytecznych wskaźników, m.in. informujących o czasie trwania tej operacji oraz o tym, czy zakończyła się sukcesem. Zbieranie danych odbywa się regularnie, zwykle w przedziale czasu od 10 do 60 sekund dla każdego monitorowanego systemu.

Pobieranie kontra przekazywanie

Prometheus to system bazujący na pobieraniu danych. Decyduje o tym, kiedy i co pobrać, bazując na konfiguracji. Istnieją również systemy bazujące na przekazywaniu, w których to monitorowany system decyduje, czy może być monitorowany i jak często.

W internecie toczone są żarliwe debaty dotyczące tych dwóch podejść projektowych, które często są podobne do debat toczonych między zwolennikami edytorów Vim i Emacs. Warto w tym miejsc wyjaśnić, że każde z tych podejść ma swoje wady i zalety. Ostatecznie nie ma znaczenia, które z nich będzie wykorzystane.

Jako użytkownik systemu Prometheus musisz zrozumieć, że mechanizm pobierania danych został wbudowany w program i podejmowanie prób mających na celu użycie podejścia bazującego na przekazywaniu danych będzie w najlepszym razie nierozsądne.

[10] Znacznik EC2 `Name` to wyświetlana nazwa egzemplarza EC2 w konsoli internetowej EC2.

Pamięć masowa

Prometheus przechowuje dane lokalnie w niestandardowej bazie danych. Zapewnienie niezawodności działania systemowi rozproszonemu jest wyzwaniem, więc Prometheus nie próbuje stosować żadnych form klastrowania. Pomijając kwestie niezawodności, to również ułatwia działanie systemu Prometheus.

Na przestrzeni lat pamięć masowa była wielokrotnie zmieniana, zaś wydanie Prometheus 2.0 posiada pamięć masową w trzeciej postaci. Ten system pamięci masowej jest w stanie obsłużyć pobieranie milionów próbek na sekundę, a tym samym pozwala, aby pojedynczy serwer systemu Prometheus był w stanie monitorować tysiące urządzeń. Zastosowany algorytm kompresji pozwala w rzeczywistych danych zmniejszyć do 1,3 bajta ilość danych w próbce. Zaleca się używanie pamięci masowej bazującej na napędach SSD, choć to nie jest ściśle wymagane.

Panele sterowania

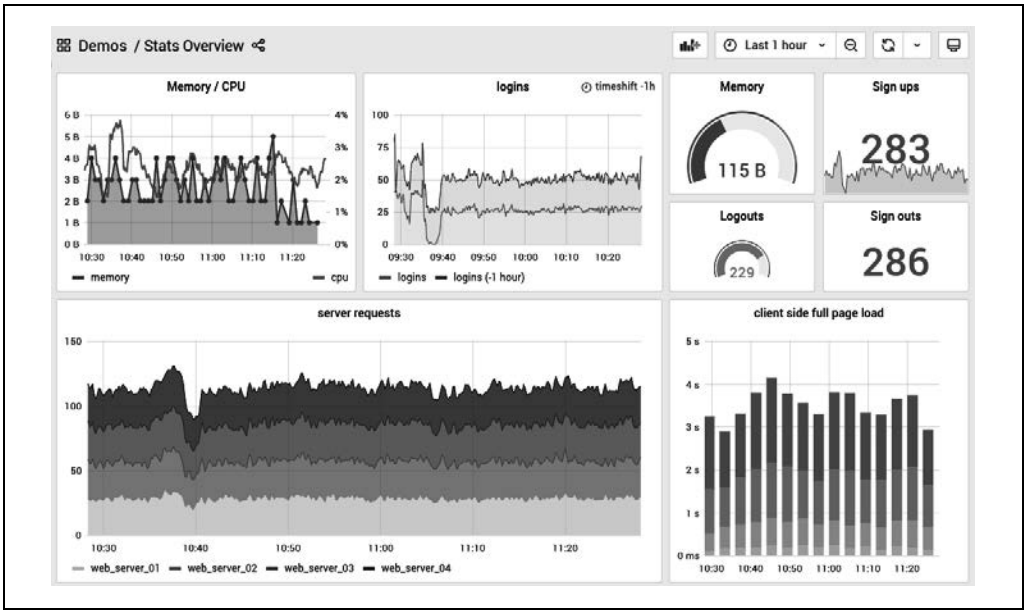
Prometheus posiada pewną liczbę API HTTP pozwalających na zarówno żądanie niezmodyfikowanych danych, jak i analizowanie zapytań PromQL. Te API można wykorzystać do generowania wykresów i paneli sterowania. Standardowo Prometheus oferuje tzw. **przeglądarkę wyrażeń** korzystającą z tych API oraz odpowiednią do wykonywania zapytań tymczasowych i eksploracji danych, choć to nie jest ogólny system paneli sterowania.

Na potrzeby zadań związanych z panelami sterowania zaleca się użycie produktu o nazwie Grafana. Oferuje on wiele funkcjonalności, np. oficjalną obsługę systemu Prometheus jako źródła danych. Grafana potrafi wygenerować wiele różnych paneli sterowania, takich jak pokazane na rysunku 1.2. Grafana obsługuje komunikację z wieloma serwerami Pythona, nawet z poziomu pojedynczego panelu sterowania.

Reguły rejestrowania i ostrzeżeń

Wprawdzie PromQL i silnik pamięci masowej zapewniają potężne możliwości i są efektywne, to jednak agregacja „w locie" wskaźników pochodzących z tysięcy urządzeń w trakcie każdego zadania generowania wykresu może spowodować, że ta operacja stanie się nieco powolna. Reguły rejestrowania pozwalają na regularne wykonywanie wyrażeń PromQL oraz umieszczanie ich wyników w silniku pamięci masowej.

Reguły ostrzeżeń to inna postać reguł rejestrowania. One również pozwalają na regularne wykonywanie wyrażeń PromQL, a ich wyniki stają się ostrzeżeniami, które następnie są przekazywane to tzw. **menedżera ostrzeżeń Alertmanager**.



Rysunek 1.2. Panel sterowania wygenerowany przez produkt Grafana (https://play.grafana.org/d/cL5pLH7Wz/stats-overview?theme=light&orgId=1)

Zarządzanie ostrzeżeniami

Menedżer ostrzeżeń Alertmanager otrzymuje ostrzeżenia z serwera systemu Prometheus i zamienia je na powiadomienia. Powiadomienia mogą obejmować wiadomości e-mail, aplikacje czatu, takie jak Slack, a także wiadomości dla usługi, takiej jak PagerDuty.

Działanie menedżera ostrzeżeń Alertmanager jest nieco bardziej złożone niż jedynie bezwarunkowa zamiana ostrzeżenia na powiadomienie w ramach systemu typu „jeden do jednego". Powiązane ze sobą ostrzeżenia mogą być agregowane na postać pojedynczego powiadomienia, co tym samym ograniczy je w celu zmniejszenia burzy pagerów[11], zaś dla poszczególnych zespołów mogą być skonfigurowane różne trasy i dane wyjściowe powiadomień. Powiadomienia można również wyciszać, np. w celu odłożenia na bok informacji o problemie, o istnieniu którego wiadomo i do usunięcia którego została już zaplanowana obsługa techniczna.

Rola menedżera ostrzeżeń Alertmanager kończy się na przekazaniu powiadomień. W celu zarządzania odpowiedzią człowieka na incydent należy używać usług takich jak PagerDuty bądź systemu bazującego na zgłoszeniach.

[11] Wspomniany **pager** to urządzenie pozwalające na przekazywanie powiadomień inżynierowi, od którego oczekuje się, że sprawdzi dane zgłoszenie. Wprawdzie istnieje możliwość przekazania zgłoszenia przez tradycyjny pager, ale obecnie prawdopodobnie zostanie użyta wiadomość SMS bądź połączenie głosowe. Burza pagerów to sytuacja, w której otrzymujesz dużą liczbę powiadomień w krótkich odstępach czasu.

Konfiguracja ostrzeżeń i ich poziomów progowych odbywa się w oprogramowaniu Prometheus, a nie w menedżerze ostrzeżeń Alertmanager.

Długoterminowa pamięć masowa

Skoro Prometheus przechowuje dane jedynie w urządzeniu lokalnym, istnieje ograniczenie dotyczące ilości miejsca na dysku dostępnego w tym urządzeniu[12]. Wprawdzie najczęściej interesują Cię dane mniej więcej z ostatniego dnia, ale na potrzeby długoterminowego planowania pojemności pożądany będzie dłuższy okres przechowywania danych.

Prometheus nie oferuje rozwiązania klastrowanego w zakresie przechowywania danych w wielu urządzeniach, choć istnieją API zdalnego odczytu i zapisu pozwalające na wykorzystanie innych systemów w podanej roli. Umożliwiają one wykonywanie zapytań PromQL do danych zarówno lokalnych, jak i zdalnych.

Czym Prometheus nie jest?

Skoro już wiesz, jak Prometheus wpasowuje się w szerszy obszar monitorowania, i znasz jego podstawowe komponenty, warto zapoznać się z przykładowymi przypadkami użycia, w których Prometheus nie okazuje się najlepszym wyborem.

Prometheus to system bazujący na wskaźnikach, a tym samym nie nadaje się do przechowywania dzienników zdarzeń lub poszczególnych zdarzeń. Nie będzie również najlepszym wyborem w przypadku danych o dużej liczności, takich jak adresy e-mail i nazwy użytkowników.

Prometheus został opracowany na potrzeby monitorowania operacyjnego, gdzie małe niedokładności i stan wyścigu ze względu na czynniki takie jak algorytmy szeregowania procesora i nieudane zebranie danych są codziennością. Używanie systemu Prometheus wiąże się z pewnymi kompromisami. Oprogramowanie woli dostarczać dane poprawne w 99,9% niż przerwać monitorowanie w oczekiwaniu na otrzymanie doskonałych danych. Dlatego też należy zachować ostrożność podczas używania systemu Prometheus w przypadkach obejmujących pracę z pieniędzmi lub systemami rozliczeniowymi.

W następnym rozdziale wyjaśnimy, jak można uruchamiać system Prometheus i przeprowadzać za jego pomocą proste monitorowanie.

[12] Jednakże nowoczesne urządzenia mogą lokalnie przechowywać ogromne ilości danych, a tym samym może nie zachodzić potrzeba używania oddzielnych klastrowanych systemów pamięci masowej.

Skorowidz

E

F

G

H

I

J

K

L

M

N

O

P

R

S

Ś

T

U

W

Z

Ź

Notatki

W tej książce znajdziesz bezcenne wskazówki dotyczące wdrażania serwera Prometheus i jego używania w rzeczywistych rozwiązaniach.

Julius Volz, współtwórca oprogramowania Prometheus

Monitorowanie wydajności systemu jest ważnym zadaniem i nie polega tylko na obserwacji pracy procesora. Należy orientować się pod jakim obciążeniem pracuje baza danych, czy przepustowość urządzeń sieciowych jest wystarczająca i jaki jest koszt niepełnego użycia bufora. Warto też wiedzieć, czy stopień wykorzystania droższych elementów uzasadnia ich zastosowanie i utrzymywanie.

To drugie, zaktualizowane wydanie przewodnika po systemie Prometheus, pozwoli zrozumieć, dlaczego ten system open source w ostatnich latach wciąż zyskuje na popularności. Znajdziesz w nim wyczerpujące wprowadzenie do tego oprogramowania, a także wskazówki dotyczące monitorowania aplikacji i infrastruktury, tworzenia wykresów, przekazywania ostrzeżeń, bezpośredniej instrumentacji kodu i pobierania wskaźników pochodzących z systemów zewnętrznych. Zrozumiesz zasady konfiguracji systemu Prometheus, komponentu Node Exporter i menedżera ostrzeżeń Alertmanager. Zapoznasz się też z nowymi funkcjonalnościami języka PromQL. Dokładnie zaprezentowano tu również zagadnienia bezpieczeństwa po stronie serwera, w tym mechanizm TLS i uwierzytelniania podstawowego.

W książce między innymi:

- czym jest Prometheus i jak wygląda jego architektura
- monitorowanie infrastruktury za pomocą komponentów takich jak Node Exporter
- instrumentacja w kodzie aplikacji
- tworzenie paneli sterowania za pomocą Grafany
- współpraca oprogramowania Prometheus i Kubernetes

Julien Pivotto jest współtwórcą serwera Prometheus i ekosystemu CNCF. Zajmuje się tym projektem od 2017 roku.

Brian Brazil jest programistą systemu Prometheus. Słynie z głębokiej znajomości tego oprogramowania.

Dzięki najlepszym praktykom i wskazówkom dotyczącym instrumentacji w kodzie ta książka pomoże Ci w niezawodnym monitorowaniu usług!

TJ Hoplock, starszy inżynier monitorowania, NS1

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-289-0529-0
9 788328 905290
Cena: 89,00 zł