Monitorowanie i obserwowalność systemów rozproszonych

mgr inż. Jakub Woźniak

Politechnika Poznańska
Wydział Informatyki i Telekomunikacji

Wprowadzenie do ekosystemu obserwowalności

- **Kontekst**: Ewolucja od monolitów do mikroserwisów i infrastruktury efemerycznej.
- **Problem**: Tradycyjny monitoring nie radzi sobie z "nieznanymi niewiadomymi".
- **Cele edukacyjne**:
  - Zrozumienie różnicy: Monitoring vs. Observability.
  - Poznanie Trzech Filarów (Metrics, Logs, Traces).
  - Standardy (OpenTelemetry) i praktyki (SRE, Chaos Engineering).

Monitoring a Observability – Ewolucja paradygmatu

Monitoring (Działanie)

- "Czy system działa?"
- Znane niewiadome (Known Unknowns).
- Reaktywny, oparty na progach.

Observability (Właściwość)

- "Dlaczego to się dzieje?"
- Nieznane niewiadome (Unknown Unknowns).
- Proaktywne badanie interakcji.

Definicja

Observability to zdolność do wnioskowania o stanie wewnętrznym systemu na podstawie jego zewnętrznej telemetrii.

Trzy Filary Obserwowalności – Architektura danych

- **Metryki (Metrics)**: Dane liczbowe w czasie (agregowalne). Służą do wykrywania trendów i alertowania.
- **Logi (Logs)**: Niezmienne rekordy zdarzeń. Dostarczają detali o błędach (granularność).
- **Ślady (Traces)**: Ścieżka żądania przez usługi. Lokalizacja wąskich gardeł latencji.

Korelacja

Kluczem jest powiązanie danych: TraceID w logach i kontekst w metrykach.

Metryki – Typy i Golden Signals

- **Typy instrumentów (OpenTelemetry)**: Counter (licznik), Gauge (miernik), Histogram.
- **The Four Golden Signals (Google SRE)**:
  1. **Latency**: Czas obsługi żądania.
  2. **Traffic**: Poziom popytu na system.
  3. **Errors**: Częstotliwość błędów.
  4. **Saturation**: Stopień wykorzystania zasobów.

Uwaga: unikaj średniej, operuj na percentylach (p95, p99).

Ekosystem Prometheus – Architektura Pull

- **Model Pull**: Serwer pobiera (scrapes) dane z endpointów HTTP (/metrics).
- **TSDB**: Lokalna baza szeregów czasowych zoptymalizowana pod zapis.
- **Komponenty**:
  - **Exporters**: Translatory dla MySQL, Node, itp.
  - **Pushgateway**: Dla zadań krótkożyjących.
  - **Service Discovery**: Automatyczne wykrywanie targetów w K8s.

PromQL i AlertManager – Logika operacyjna

- **PromQL**: Funkcyjny język zapytań (agregacja, predykcja).
- **AlertManager**: Zarządzanie alertami z Prometheusa.
  - **Grouping**: Łączenie alertów w jeden komunikat.
  - **Inhibition**: Wyciszanie alertów podrzędnych.
  - **Silencing**: Wyciszanie na czas prac konserwacyjnych.

Zarządzanie logami – Stos ELK vs. EFK

- **Elasticsearch**: Baza NoSQL do wyszukiwania pełnotekstowego.
- **Kibana**: Wizualizacja i dashboardy.[22, 23]
- **Logstash (ELK)**: Potężny silnik ETL (JVM), duży narzut zasobów.
- **Fluentd/Fluent Bit (EFK)**: Lekkie kolektory (CNCF), natywne dla K8s.

Fluent Bit jest zalecany jako agent brzegowy ze względu na minimalne zużycie RAM.]

Distributed tracing – Jaeger i Zipkin

- **Cel**: Śledzenie żądań między granicami serwisów.
- **Zipkin**: Pionier (Twitter), prosty w konfiguracji (Java).
- **Jaeger**: Nowoczesny standard (Uber/CNCF), wysoka skalowalność (Go).
- **Pojęcia**:
  - **Span**: Jednostka pracy.
  - **Context Propagation**: Przekazywanie nagłówków.
  - **Sampling**: Decyzja o zapisaniu śladu (np. 1% ruchu).

Rachunki związane z obserwowalnością mogą łatwo przewyższyć koszt chmury

OpenTelemetry (OTel) – Unifikacja telemetrii

- **Idea**: Jeden standard dla metryk, logów i śladów.
- **OTel Collector**:
  - *Receivers*: OTLP, Jaeger, Prometheus.
  - *Processors*: Filtrowanie, batching, PII redaction.
  - *Exporters*: Wysyłka do wielu backendów naraz.

OTel rozwiązuje problem uzależnienia od dostawcy (Vendor Lock-in).

SRE practices – SLI, SLO i Error Budgets

- **SLI**: $SLI = \frac{\text{Liczba poprawnych zdarzeń}}{\text{Suma zdarzeń}}$.
- **SLO**: Cel (np. 99,9% dostępności).
- **Error Budget**: $100\% - SLO$.

Zasada Error Budget

Jeśli budżet błędów się wyczerpie, priorytetem staje się stabilność (wstrzymanie nowych funkcji).

AIOps – Machine Learning w monitorowaniu

- **Anomaly Detection**: Dynamiczne progi zamiast statycznych.
- **Noise Reduction**: Grupowanie tysięcy alertów w jeden incydent.
- **Predictive Analytics**: Przewidywanie awarii (np. zapełnienie dysku).
- **Root Cause Analysis (RCA)**: Automatyczne wskazywanie winnego.

Chaos Engineering – Testowanie niezawodności

- **Steady State**: Definicja stanu normalnego.
- **Hypothesis**: "Jeśli wyłączymy instancję bazy, system obsłuży ruch przez replikę".
- **Minimalizacja Blast Radius**: Ograniczenie wpływu na użytkowników.
- **Narzędzia**: Chaos Monkey, Gremlin, Litmus.

Design monitoring strategy dla systemu e-commerce

- **Kluczowe procesy**: Checkout, Cart, Payment.
- **Hierarchia monitoringu**:
  1. Warstwa biznesowa: Sprzedaż na sekundę.
  2. Warstwa aplikacji: Latencja p95 płatności ($<$300ms).
  3. Warstwa infrastruktury: Zdrowie kontenerów K8s.

Incident response i Postmortem analysis

- **Incident Lifecycle**: Detection → Triage → Mitigation → Resolution.
- **Blameless Postmortem**:
  - Skupienie na systemie, nie na człowieku.
  - Cel: Wypracowanie Action Items zapobiegających nawrotom.
  - Timeline i Root Cause Analysis.