# Część 2: Backend — APM (.NET/Node): SLO, latency p95/p99, error rate, dependencies

Scenariusz prowadzącego (20–30 min) — stack: .NET Core + Angular + Node.js

## 0) Cel i efekty

• Zrozumieć, jak w Kibanie czytać metryki APM dla usług .NET/Node i szybko lokalizować źródła degradacji.

• Umieć pracować z p95/p99 latency, error rate, throughput oraz breakdownami per endpoint.

• Korelować problemy z zależnościami (DB/cache/HTTP) i przechodzić do metryk infrastruktury (Metricbeat).

• Zdefiniować SLO (latency/availability) oraz związać je z alertami.

## 1) Przygotowanie „dzień wcześniej” (poza sesją)

• Instrumentacja APM: agent .NET dla API .NET Core oraz agent Node.js dla usług Node. Ustal `service.name`, `service.environment`, `service.version`.

• Tracing rozproszony: propagacja W3C Trace Context (`traceparent`, `tracestate`) przez reverse proxy (IIS/Nginx/Traefik) oraz CORS dopuszczający nagłówki.

• Runtime metrics (opcjonalnie): włącz metryki środowiska uruchomieniowego (.NET GC, wątki; Node event loop) – zgodnie z dokumentacją agenta.

• Sampling: ustaw sampling (head/tail) tak, by w godzinach szczytu mieć reprezentatywne próbki (np. 10–20%), a błędy zawsze pełne.

• Saved Objects/Queries: zaimportowany NDJSON z poprzedniej części (Dashboard + Saved Queries).

• Zakładka SLO w Observability: upewnij się, że możesz tworzyć SLO typu APM latency/availability.

## 2) Start sesji – ustawienia w Kibanie

• Zakres czasu: Last 24 hours (lub 7 days dla analizy trendu). Włącz „Compare to previous period”.

• Filtry globalne: `service.environment : "prod"` (lub `staging`). W razie potrzeby dodaj `service.name` wybranej usługi.

• Auto-refresh: 1–5 min na czas pokazu.

## 3) Widok APM — orientacja w UI

• Services: lista usług z Throughput, Latency, Error rate; sortowanie i filtracja.

• Service overview: szybki przegląd trendów (latency/error/throughput) i podsumowanie endpoints (transaction groups).

• Transactions: szczegóły endpointów (histogram duration, p95/p99, próbki trace, „waterfall” ze spanami).

• Dependencies: zewnętrzne zależności (DB, Redis, HTTP) z ich latency i error rate.

• Errors: grupy wyjątków z wystąpieniami, stacktrace i release (service.version).

• Service map: powiązania usług; identyfikacja krawędzi z wysoką latencją/błędem.

## 4) Scenariusz pokazu — krok po kroku

• Krok A — Services: sortuj po „Error rate”, potem po „Latency”. Wybierz usługę krytyczną (np. `api-orders`).

• Krok B — Service overview: oceń trend p95/p99, error rate i throughput; porównaj do poprzedniego okresu (Compare).

• Krok C — Transactions: wybierz najwolniejszy endpoint. Obejrzyj histogram (rozrzut), mediana vs p95/p99, następnie otwórz przykładowy trace (View sample).

• Krok D — Trace waterfall: prześledź spany DB/HTTP/Redis; znajdź najdłuższy span. Sprawdź atrybuty (np. SQL, URL, kod HTTP) i czy są retry.

• Krok E — Dependencies: zobacz, czy to DB/cache z wysoką latencją albo HTTP do innej usługi. Jeśli target też jest instrumentowany – kliknij do niej. Jeśli nie — przejdź do metryk Metricbeat (MSSQL/Redis/Nginx).

• Krok F — Errors: sprawdź nowe grupy wyjątków, ich częstotliwość i wpływ. Skoreluj z konkretnymi releases (service.version).

• Krok G — Service map: oceń problematyczne krawędzie (wysoka latencja/erro-rate).

• Krok H — Przejście do Infra: jeśli podejrzewasz zasoby, otwórz Infrastructure → host/pod (CPU/memory), reverse proxy (5xx/min), DB (waits/evictions).

## 5) p95/p99, error rate, throughput — interpretacja

• p95 vs p99: p95 pokazuje typowe górne opóźnienie; p99 odsłania „długi ogon”. Wzrost p99 bez wzrostu p95 często wskazuje na sporadyczne problemy (np. blokady DB).

• Error rate: patrz zarówno wartości względne (%) jak i bezwzględne (liczba błędów) — przy niskim throughputcie nawet 2 błędy dadzą wysoki %.

• Throughput: spadek przepływu przy stałej latencji może oznaczać ograniczenia zasobowe (backpressure) lub błędy powodujące retriable failures.

## 6) Dependencies — jak diagnozować

• DB (MSSQL/Postgres/Mongo): rosnące czasy zapytań, locki/waits → koreluj z metrykami Metricbeat (mssql/postgresql/mongodb).

• Redis: rosnące latency/evictions → sprawdź `redis` w Metricbeat (pamięć, eviction policy).

• HTTP zewnętrzne: kody 5xx/429, timeouty; sprawdź retry i polityki circuit breaker.

• Nieinstrumentowane zależności: przechodź do Logs/Infra po `destination.address` i metrykach reverse proxy (IIS/Nginx/Traefik).

## 7) SLO — definicje i konfiguracja

• SLI (availability): 1 − (failed transactions / all transactions). Wybierz „APM availability” i wskaż transakcje (np. typ=request).

• SLI (latency): odsetek transakcji w czasie < X ms (np. 95% < 300 ms). Wybierz „APM latency” i ustaw docelowy próg.

• SLO: celu i budżet błędu (np. 99.9% miesięcznie). Skonfiguruj alerty: naruszenie budżetu i wczesne ostrzeganie (burn rate).

## 8) Saved Queries — KQL dla APM

• APM — Transakcje HTTP (24h)

processor.event : "transaction" and transaction.type : "request" and @timestamp >= now-24h

• APM — Błędy (24h)

processor.event : "error" and @timestamp >= now-24h

• APM — Najwolniejsze endpointy (>1s)

processor.event : "transaction" and transaction.type : "request" and transaction.duration.us >= 1000000 and @timestamp >= now-24h

• APM — Usługi z wysokim error rate

processor.event : "error" and @timestamp >= now-24h | stats count() by service.name | sort desc

## 9) Alerty — zestaw startowy

• APM: Error rate > 2% przez 5 min (na krytycznych usługach).

• APM: p95 latency > 1 s przez 5 min.

• APM: Nowa grupa błędów (spike) — alert na liczbę unikalnych grup w 15 min.

• Dependencies: Wzrost latency na DB/Redis/HTTP o >50% vs poprzedni okres.

• SLO: Naruszenie progu budżetu błędu (burn rate) — tryb szybki i wolny (multi-window).

## 10) Lab — ćwiczenia praktyczne

• Ćw. 1: Znajdź endpoint o najwyższym p95; wyświetl trace i zidentyfikuj najdłuższy span.

• Ćw. 2: W Dependencies znajdź zależność z najwyższą latencją; wskaż, czy to DB/Redis/HTTP i jaka metryka w Metricbeat to potwierdza.

• Ćw. 3: Zbuduj SLO „APM latency” dla krytycznego endpointu (95% < 300 ms, 30 dni) i skonfiguruj alert wczesnego wypalania budżetu.

• Ćw. 4: Utwórz alert „APM Error rate” i zasymuluj błąd; sprawdź, czy pojawił się alert i utwórz Case.

## 11) Pułapki i dobre praktyki

• Sampling: przy zbyt agresywnym samplingu p99 będzie niestabilne — dla krytycznych tras rozważ tail-based sampling lub wyższy sampling.

• Health-checki: odfiltruj transakcje health-check (np. `/health`) przy obliczaniu SLO/alertów, by nie fałszowały metryk.

• Nazewnictwo: trzymaj stabilne `service.name` i `transaction.name` (np. `/api/orders/{id}` jako szablon, nie konkretne ID).

• Wersjonowanie: używaj `service.version` — szybciej wykryjesz regresję po deployu.

• Korelacja z logami: łącz trace.id z logami (ECS `trace.id`) dla pełnego kontekstu incydentu.

## 12) Mini-ściąga dla prowadzącego

• Services → sort by Error rate/Latency → wybierz usługę.

• Service overview → porównaj trend p95/p99 i error rate.

• Transactions → najwolniejszy endpoint → Trace waterfall.

• Dependencies → winowajca DB/Redis/HTTP? → przejście do Metricbeat.

• Errors → nowe grupy wyjątków i impact.

• Zbuduj/Sprawdź SLO i alerty.