Kibana — wykorzystanie danych z APM (.NET Core, instrumentacja w kodzie)

Propozycja dokumentu: APM UI + Discover/Lens, KQL, alerty i korelacje dla aplikacji .NET Core z agentem APM dodanym jako paczka w kodzie.  
*Data: 2025-08-21 07:34*

# 1) Cel dokumentu

* Pokazać co i jak wyciągać z danych APM w Kibanie, gdy .NET Core jest zainstrumentowany paczką w kodzie.
* Dostarczyć gotowe widoki APM, zapytania KQL, przepisy Lens i alerty do szybkiego użycia.
* Dodatkowo: minimalny przykład konfiguracji agenta w .NET (Program.cs + appsettings.json).

# 2) Gdzie są dane i jak je przeglądać

* Aplikacja: Observability → APM (Services/Transactions/Traces/Errors/Dependencies/Service map).
* Data view (dla Discover/Lens): apm-\* (zwykle aliasy do metrics-apm\*, logs-apm\*, traces-apm\*).
* Najważniejsze pola: service.name, service.environment, service.version, transaction.\* (duration.us, name, type, result), span.\* (type/subtype, duration.us, destination.\*), error.\* (exception, log), trace.id, transaction.id, user.\*, labels.\*.

# 3) Szybki start w Kibanie (APM UI)

* Services: lista usług z p50/p95/p99 latency, throughput i error rate — wybierz swoją usługę .NET.
* Transactions: podgląd HTTP route/queue/background; wykres latencji i rozkład wyników (2xx/4xx/5xx).
* Traces: widoki end‑to‑end (frontend→backend→DB) z rozbiciem na spany i czasy.
* Errors: grupy błędów (stacktrace, fingerprint), trend i przykłady logów błędów.
* Dependencies: zewnętrzne usługi (DB, HTTP), ich czas i error rate; Service map: relacje między usługami.

# 4) Minimalna konfiguracja agenta w kodzie (.NET Core)

Dodaj paczki NuGet (warianty zależnie od preferencji/wersji agenta):

# Package Manager Console  
Install-Package Elastic.Apm.AspNetCore  
Install-Package Elastic.Apm.SqlClient  
# (opcjonalnie) Redis/Mongo/HttpClient instrumentations

Program.cs — wczesna inicjalizacja middlewara APM:

using Elastic.Apm.AspNetCore;  
// ...  
  
var builder = WebApplication.CreateBuilder(args);  
builder.Services.AddControllers();  
  
var app = builder.Build();  
  
// Wariant A: UseElasticApm (nowszy)  
app.UseElasticApm(builder.Configuration);  
  
// Wariant B: UseAllElasticApm (starszy/zbiorczy, jeśli używasz meta‑paczki)  
/\*  
using Elastic.Apm.NetCoreAll;  
app.UseAllElasticApm(builder.Configuration);  
\*/  
  
app.MapControllers();  
app.Run();

Konfiguracja w appsettings.json (lub zmienne środowiskowe):

{  
 "ElasticApm": {  
 "ServerUrls": "https://<APM\_SERVER>:8200",  
 "ServiceName": "my-api-dotnet",  
 "Environment": "prod",  
 "SecretToken": "<token-if-any>",  
 "TransactionSampleRate": 0.5,  
 "CaptureHeaders": true,  
 "CaptureBody": "off" // włącz wg polityki prywatności: "all" / "errors"  
 }  
}

Zmienne środowiskowe (alternatywnie):

ELASTIC\_APM\_SERVER\_URLS=https://<APM\_SERVER>:8200  
ELASTIC\_APM\_SERVICE\_NAME=my-api-dotnet  
ELASTIC\_APM\_ENVIRONMENT=prod  
ELASTIC\_APM\_TRANSACTION\_SAMPLE\_RATE=0.5

# 5) Typowe pytania, na które odpowie Kibana (APM)

* Jakie endpointy są najwolniejsze (p95/p99)? Które transakcje mają najgorszy throughput vs latency?
* Jaki jest error rate i które wyjątki dominują?
* Które zależności (DB/HTTP) są wąskim gardłem?
* Czy regresja pojawiła się po nowym wydaniu (service.version)?
* Jak wygląda ślad (trace) żądania end‑to‑end, łącznie z RUM/backend/DB?

# 6) Przepisy „Lens” (na danych APM) — 30–90 s

A. Trend p95 latency transakcji HTTP

* Filter: processor.event:"transaction" and transaction.type:"request" and service.name:"my-api-dotnet"; Metric: 95th percentile of transaction.duration.us (→ ms).

B. Error rate usługi

* Filter: processor.event:"error" and service.name:"my-api-dotnet"; Metric: Count; Add formula: count(errors) / count(transactions) (lub użyj gotowego wskaźnika w APM UI).

C. Top powolne spany DB

* Filter: processor.event:"span" and span.type:"db" and service.name:"my-api-dotnet"; Metric: 95th percentile of span.duration.us; Break down by: span.subtype (mssql/postgresql).

D. Czas własny (self time) — identyfikacja wąskich gardeł w kodzie

* W APM UI w szczegółach transakcji zaznacz panel „Span self time” i sortuj po największych wartościach; w Discovery można zbliżyć się, używając pól `span.duration.us` oraz typów spanów.

# 7) KQL — gotowe „cookbook” (Discover / Lens)

service.name : "my-api-dotnet" and processor.event : "transaction" and transaction.type : "request" and transaction.duration.us >= 1000000 # transakcje ≥ 1s

service.name : "my-api-dotnet" and processor.event : "error" # błędy usługi

service.name : "my-api-dotnet" and processor.event : "span" and span.type : "db" # spany DB

service.name : "my-api-dotnet" and processor.event : "transaction" and http.response.status\_code >= 500 # 5xx

labels.release : "2025.08.15" and processor.event : "transaction" # porównanie wersji/releasów

trace.id : "<trace-id>" # korelacja śladu

# 8) Alerty — gotowy pakiet reguł (APM)

A. Latency threshold (p95)

* Zakres: service.name:"my-api-dotnet", transaction.type:"request"; Warunek: p95(transaction.duration) > 1000 ms przez 5 min; Group by: transaction.name.

B. Error rate > X%

* Warunek: (errors / transactions) > 5% przez 5–10 min; Group by: service.name, transaction.name.

C. Spike w zależności (DB/HTTP)

* Filter: processor.event:"span" and span.type:("db" or "external"); Warunek: p95(span.duration) > próg; Group by: span.subtype, destination.service.name/url.domain.

D. „No data” (brak metryk/trace)

* Typ: Less than 1 match; KQL: service.name:"my-api-dotnet" and processor.event:"transaction"; Okno: 5–10 min.

E. Error logs ↔ APM (jeśli korelacja z logami)

* Reguła KQL na `processor.event:"error"` + integracja z kanałem powiadomień; alternatywnie reguły w APM UI dla grup błędów.

# 9) Korelacje (APM ↔ inne źródła)

* Logi (Filebeat/Serilog): dodaj korelację przez `trace.id`/`transaction.id` w logach — wtedy Discover pokazuje logi danej transakcji.
* RUM (frontend): gdy włączone distributed tracing (W3C), w Traces zobaczysz ścieżkę od przeglądarki do .NET i dalej.
* Metricbeat: CPU/RAM/Disk hosta w pikach latency — korelacja po host.name / service.node.name.
* Heartbeat: spadki dostępności vs error rate/latency w APM.
* Packetbeat: piki `event.duration` HTTP i błędy 5xx na ścieżkach koreluj z trace’ami APM.

# 10) Raportowanie i współdzielenie

* Udostępniaj linki do widoków APM z predefiniowanymi filtrami (service.name, environment, transaction.name).
* Harmonogram PDF (Reporting) dla kluczowych dashboardów i/lub paneli Lens z danych apm-\*.
* Twórz Saved Searches (np. „APM — slow transactions ≥ 1s”).

# 11) Dobre praktyki (agent i analiza)

* Ustaw `service.name`, `service.environment`, `service.version` (np. z CI/CD) — kluczowe dla porównań wydań.
* Sampling (`TransactionSampleRate`): zacznij od 0.3–0.5; na produkcji wyższy kosztuje więcej zasobów/danych.
* Dodaj `labels.\*` z metadanymi biznesowymi (np. tenant, region) — ułatwia filtry i SLI/SLO.
* Włącz instrumentacje DB/HTTP; dodaj własne spany w krytycznych miejscach (np. kolejki).
* Korelacja z logami: w Serilog użyj `WithElasticApmCorrelationInfo()` i wypisuj trace.id/transaction.id do logów.
* Prywatność: rozważ `CaptureBody` i `SanitizeFieldNames` zgodnie z polityką; nie przesyłaj wrażliwych danych.

# 12) „Starter pack” (do importu / odtworzenia)

Saved Searches:

APM — Slow HTTP ≥ 1s: service.name:"my-api-dotnet" and processor.event:"transaction" and transaction.type:"request" and transaction.duration.us >= 1000000

APM — Errors (24h): service.name:"my-api-dotnet" and processor.event:"error" and @timestamp >= now-24h

Dashboardy Lens (z apm-\*):

* p95 latency per transaction.name (filter service.name).
* Error rate (count errors / count transactions).
* Top DB spans by p95 (span.subtype).

Alerty:

* APM\_LATENCY\_P95\_THRESHOLD, APM\_ERROR\_RATE, APM\_DB\_SPAN\_SLOW, APM\_NO\_DATA.

# 13) Aneks — korelacja z logami (Serilog przykład)

// Serilog + Elastic APM correlation  
using Serilog;  
using Elastic.Apm.SerilogEnricher;  
  
Log.Logger = new LoggerConfiguration()  
 .Enrich.WithElasticApmCorrelationInfo() // adds TraceId, TransactionId, SpanId  
 .WriteTo.Console(outputTemplate: "[{Timestamp:HH:mm:ss} {Level:u3}] (TraceId={TraceId} TxId={TransactionId}) {Message:lj}{NewLine}{Exception}")  
 .CreateLogger();

# 14) Aneks — przykładowy docker-compose (opcjonalnie)

environment:  
 - ELASTIC\_APM\_SERVER\_URLS=https://<APM\_SERVER>:8200  
 - ELASTIC\_APM\_SERVICE\_NAME=my-api-dotnet  
 - ELASTIC\_APM\_ENVIRONMENT=prod  
 - ELASTIC\_APM\_TRANSACTION\_SAMPLE\_RATE=0.5

# Co dalej?

Mogę przygotować pakiet Saved Objects (widoki APM, Lens, alerty) oraz szablony appsettings dla różnych środowisk i integracje z loggerem (.NET Serilog/NLog) dopasowane do Twojej organizacji.