# Metricbeat — co monitorować w stacku .NET Core + Angular + Node.js oraz związek z APM .NET i RUM Angular

## 1) MetricBeat, co może monitorować dla Stacku: .NET Core + Angular + Node.js

Jasne! Poniżej zestawiłem **co Metricbeat może monitorować** w typowym stacku **.NET Core + Angular + Node.js** – podzielone na warstwy. (Tu tylko Metricbeat; do metryk z przeglądarek i samego kodu SPA lepszy będzie APM RUM, ale to osobny agent.)

### .NET Core (API / usługi)

* System/host: `system` (CPU, RAM, dyski, procesy, sieć), Windows (PerfMon/WMI), Linux (beta).
* HTTP/frontend reverse proxy: IIS (jeśli hostujesz pod IIS), Nginx, HAProxy, Traefik.
* Aplikacja .NET – metryki aplikacyjne: Prometheus/OpenMetrics (skrapuje `/metrics` z `prometheus-net` lub OpenTelemetry Metrics), StatsD (jeśli aplikacja wysyła metryki do StatsD), HTTP (generic) (jeśli wystawisz własny JSON health/metrics).
* Uwaga: klasyczne liczniki “.NET CLR” z PerfMon dotyczą głównie .NET Framework; dla .NET Core/ASP.NET Core polecane jest eksponowanie metryk jako Prometheus/OTel i skrapowanie przez Metricbeat.
* Baza danych / cache / kolejki (jeśli używasz): MSSQL (SQL Server), PostgreSQL, MySQL/MariaDB, Redis, RabbitMQ, Kafka, MongoDB.
* Kontenery/Orkiestracja: Docker, containerd (beta), Kubernetes.
* Chmura (jeśli hostowane w cloud): Azure (Azure Monitor), AWS (CloudWatch: EC2/ALB/RDS/CloudFront itp.), GCP (Cloud Monitoring).

### Angular (SPA)

* Serwer plików statycznych / CDN / edge: Nginx, Apache, IIS, (w chmurze: AWS/CloudFront/ALB via AWS module, Azure Front Door/App Service via Azure module).
* System/host: `system`, Windows, Linux (beta) – jeśli serwujesz SPA z własnego hosta.
* SSR / middleware (jeśli masz Angular Universal na Node): patrz sekcja Node.js + Nginx/Traefik/HAProxy przed nim.
* Własne health/metrics: HTTP (generic) (JSON), Prometheus/OpenMetrics (jeśli serwer serwujący SPA/edge wystawia `/metrics`).

*Metricbeat nie zbiera „web-vitals” z przeglądarki; do TTFB/LCP/CLS użyj APM RUM.*

### Node.js (API / worker / SSR)

* Aplikacja Node: Prometheus/OpenMetrics (skrapowanie `/metrics` z `prom-client`), StatsD (Node client → Metricbeat), HTTP (generic) (custom JSON).
* System/host i procesy: `system` (w tym `process`), Linux (beta) / Windows – CPU, pamięć, I/O, sieć, proces `node`.
* Reverse proxy / ingress: Nginx, Traefik, HAProxy.
* Datastores / kolejki (typowe w Node): MongoDB, PostgreSQL, MySQL, Redis, RabbitMQ, Kafka.
* Kontenery / orkiestracja: Docker, containerd (beta), Kubernetes.
* Chmura: AWS/Azure/GCP moduły do CloudWatch/Monitor/Cloud Monitoring (np. ECS/EKS, App Service, GKE).

### Szybkie „pakiety startowe” (propozycje)

• API .NET Core na Windows pod IIS + MSSQL + Redis

* APM: .NET agent (transakcje, błędy, zależności).
* Metricbeat: `windows`/`system`, `iis`, `nginx`/`traefik`, `mssql`, `redis` (+ `prometheus` jeśli masz własne metryki).
* RUM (jeśli SPA pod tym API): do obserwacji UX i web-vitals.

• Node.js API w Docker/K8s + MongoDB + Nginx

* APM: —
* Metricbeat: `docker`/`kubernetes`, `system`, `nginx`, `mongodb`, `prometheus` (z `prom-client`), opcj. `statsd`.

• Angular SPA na Nginx (VM) + CDN w AWS

* RUM: web-vitals i błędy frontendowe.
* Metricbeat: `nginx`, `system`, `aws` (CloudFront/ALB/EC2 via CloudWatch), opcj. `http` dla health-checka.

*Chcesz, żebym z tego zrobił krótki plik DOCX z checkboxami „co włączyć” dla Twojej dokładnej architektury? Podaj, gdzie hostujesz poszczególne elementy (Windows/Linux, on-prem/cloud, kontenery).*

## 2) Związek Metricbeat ↔ APM (.NET Core) i RUM (Angular)

### Co mierzy kto (podział ról)

* APM .NET Core: transakcje i ślady rozproszone (HTTP, DB, cache), wyjątki, error rate, latency, dependency map (service → DB/queue), metryki runtime (.NET GC, wątki – zależnie od konfiguracji/OTel). To „co robi aplikacja”.
* RUM Angular (APM RUM JS): real-user metrics w przeglądarce (LCP, FID/INP, CLS, TTFB), błędy JS/AJAX, czasy nawigacji/route change, urządzenia/przeglądarki/położenie. To „co widzi użytkownik”.
* Metricbeat: infrastruktura i otoczenie aplikacji (host OS, IIS/Nginx/Traefik/HAProxy, DB: MSSQL/PostgreSQL/Redis/Mongo, kolejki: RabbitMQ/Kafka, kontenery: Docker/K8s, chmura: AWS/Azure/GCP). To „na czym to działa”.

### Jak to łączyć w Elastic (praktyka)

* Spójne identyfikatory (ECS): Ustal i trzymaj konsekwentnie: `service.name`, `service.environment`, `service.version` w APM (.NET) i RUM (Angular). Metricbeat domyślnie wypełnia `host.\*`, `container.\*`, `kubernetes.\*` i `service.type` (np. `iis`, `redis`). Jeśli chcesz łatwiej korelować, dodaj etykiety/fields odzwierciedlające środowisko (np. `labels.env=prod`) tym samym słownikiem co w APM.
* Korelacje i nawigacja: W APM przejdziesz z transakcji .NET do Service map i Dependencies; metryki dla np. MSSQL/Redis możesz porównać z modułami Metricbeat (`mssql`, `redis`). Z RUM → APM: dla wolnych widoków/tras sprawdzasz backendy w APM, a następnie Infra (Metricbeat) host/reverse proxy/DB. Uzgodnij `host.name` i (jeśli kontenery) `container.id` między APM a Metricbeat — to ułatwia pivot z transakcji do metryk hosta/poda.
* Metryki aplikacyjne przez Prometheus/OTel: Jeżeli potrzebujesz własnych KPI aplikacji: eksponuj `/metrics` (Prometheus) w .NET (`prometheus-net`, OTel Metrics) i scrapuj je modułem Prometheus/OpenMetrics w Metricbeat. Unikaj dublowania: nie skrapuj tego samego eksportera i jednocześnie nie wysyłaj tych samych metryk innym kanałem.

### Typowe zestawy „end-to-end”

* API .NET Core + IIS + MSSQL + Redis + Nginx/Traefik: APM: .NET agent (transakcje, błędy, zależności). Metricbeat: `windows`/`system`, `iis`, `nginx`/`traefik`, `mssql`, `redis` (+ `prometheus` jeśli masz własne metryki). RUM (jeśli SPA pod tym API): do obserwacji UX i web-vitals.
* Angular SPA + Nginx (VM/K8s) + CDN (cloud): RUM: web-vitals i błędy frontendowe. Metricbeat: `nginx`, `system`/`kubernetes`, moduł chmurowy (`aws`/`azure`) dla ALB/Front Door/CloudFront.

### Alertowanie — jak złożyć klocki

* SLO aplikacyjne (APM): latency p95/p99 i error rate dla kluczowych tras .NET → jeśli naruszone, „drill down” do Metricbeat (CPU throttling hosta/poda, 5xx na Nginx/IIS, kolejki DB).
* UX (RUM): wzrost LCP/INP lub skok błędów JS → sprawdź APM (czy backend zwolnił) → sprawdź Metricbeat (sieć, reverse proxy, DB).
* Warstwa danych (Metricbeat): wzrost `Page life expectancy`/`Log flush waits` (MSSQL) lub evictions w Redis → koreluj z APM (wzrost latency transakcji zależnych).

### Minimalne zasady porządkowe

* Ten sam naming dla `service.name` w APM (.NET) i RUM (Angular) oraz ta sama wartość `service.environment` (np. `prod`, `staging`).
* W Metricbeat dodaj `labels.environment`/`labels.team` tak, by filtry/dashbordy były wspólne z APM/RUM.
* Ustal jeden sposób wersjonowania (`service.version`) — przy deployach szybciej wykryjesz regresje w APM i RUM, a w Metricbeat zobaczysz ich korelaty infrastrukturalne.

*Chcesz, żebym dopiął tę sekcję do utworzonego wcześniej pliku Word i wygenerował nową wersję do pobrania?*