deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

Observabilidade com OpenTelemetry no nopCommerce

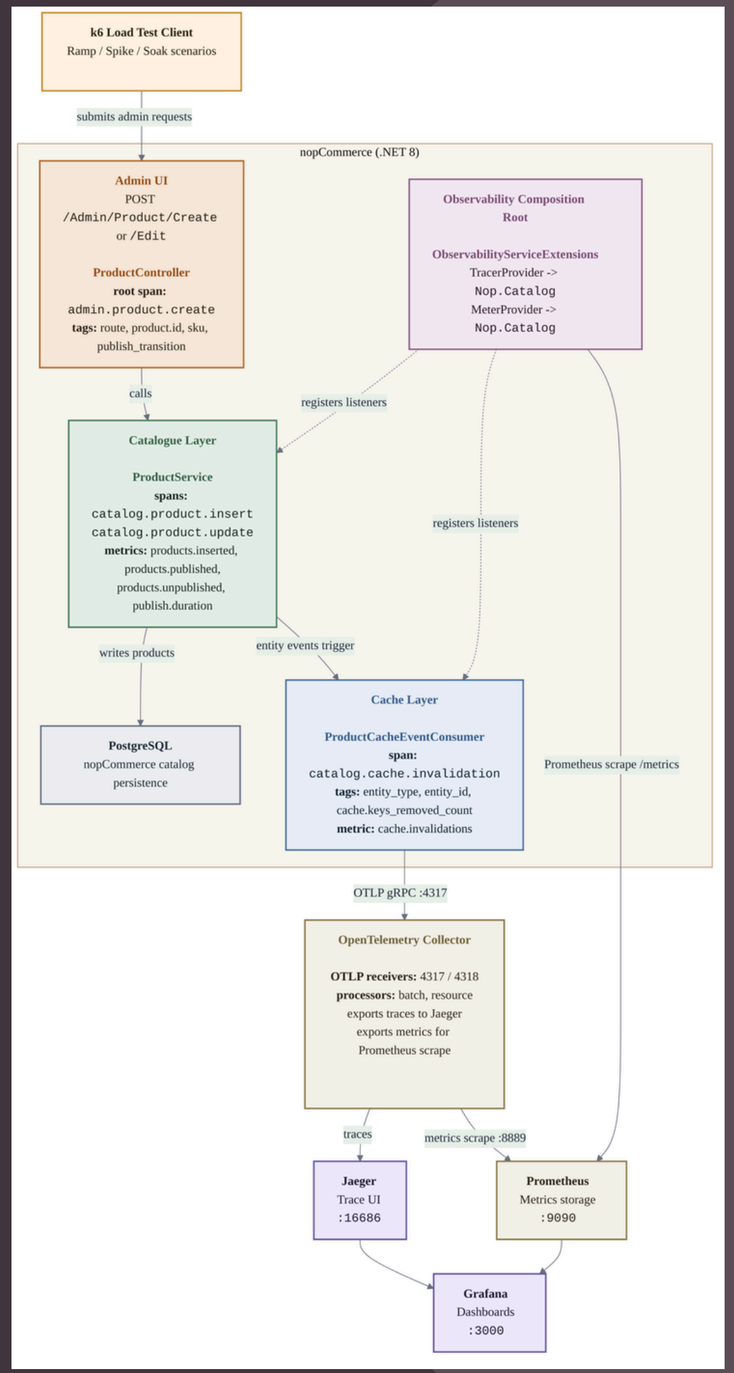
Trabalho 1: Administrador publica produto novo

Luana Carolina Reis, nº 131193
24 de março de 2026

Fluxo Escolhido e Arquitetura

Fluxo escolhido: Administrador publica produto novo

- Ponto de entrada em ProductController
- Lógica de negócio em ProductService
- Invalidação de cache através de eventos de entidade
- Exportação para Collector, Prometheus, Jaeger e Grafana



Desenho da Instrumentação

Estrutura de tracing

- admin.product.create como root span na fronteira HTTP do admin
- catalog.product.insert / catalog.product.update em ProductService
- catalog.cache.invalidation no consumer de cache

Desenho vendor-neutral

- Nop.Services usa apenas tipos de System.Diagnostics
- O SDK e os exporters de OpenTelemetry existem apenas em Nop.Web
- Isto mantém a instrumentação portável e preserva as fronteiras entre camadas

Estratégia de privacidade

- Emitir apenas tags técnicas e de negócio explicitamente selecionadas
- Excluir identidade do admin, corpo do pedido, faturação, pagamentos e campos de texto livre
- Minimizar os dados na origem antes de entrarem no pipeline de telemetry

Métricas Personalizadas e Porque Importam

Métrica	Porque importa
catalog.products.published	Confirma throughput de negócio e se os produtos estão realmente a ficar visíveis.
catalog.product..publish.duration	Deteta degradação de latência no pipeline de publish antes de surgirem falhas.
catalog.cache.invalidations	Confirma que o principal side-effect após escrita continua a acontecer.
catalog.products.inserted	Distingue criação de drafts de transições reais para estado publicado.
catalog.products.unpublished	Torna reversões e ações inesperadas do admin operacionalmente visíveis.

Tabela 1 - Métricas personalizadas criadas

Mudanças cirúrgicas no Código Existente

O objetivo foi adicionar observabilidade com disrupção mínima!

- ProductController: a root span começa na fronteira do admin
- ProductService: spans e contadores de negócio em insert / update / publish
- ProductCacheEventConsumer: emissão de trace e métrica para invalidação de cache
- ObservabilityServiceExtensions: todo o wiring do SDK isolado na composition root
- Pequeno overload com publishTransition para evitar uma query extra à base de dados só para telemetry

Intenção arquitetural

- Sem refactor alargado dos internals do nopCommerce
- O significado de negócio fica nos services, e os exporters ficam na host layer

Assessment Arquitetural do nopCommerce

O que ajudou

- Separação clara entre presentation, services, data e core
- IProductService oferece uma boa fronteira de negócio para domain spans
- Eventos de repositório e cache consumers expõem side-effects pós-escrita de forma limpa
- A composition root em Nop.Web facilita isolar o wiring do SDK

O que dificultou

- As fronteiras dos fluxos num monólito nem sempre são explícitas
- O event publishing genérico esconde o grafo real de eventos
- Side-effects importantes aparecem de forma indireta após a persistência

Demonstração e Evidência

Sequência da demonstração

1. Mostrar o dashboard de Grafana pronto a receber sinal
2. Executar um cenário de k6 já preparado
3. Em Prometheus, mostrar o total de publishes e o p95
4. Em Jaeger, abrir um trace admin.product.create e mostrar a cadeia de 3 spans

Scenario	Publishes	Publish p95	Failures
Ramp	717	538 ms	0%
Spike	574	2.09 s	0%
Soak	1573	466 ms	0%

Tabela 2 - Resultados observados durante 3 load tests diferentes

Resumo

- O fluxo está observável de ponta a ponta
- O dashboard reage de forma útil sob carga
- A plataforma foi instrumentada de forma cirúrgica, sem um refactor alargado

OBRIGADA pela atenção!

Questões?