SREcon26 Americas

Resilient Observability at the Retail Edge

A Lightweight, Scalable, and Cost-Efficient Framework

Prakash Velusamy · Principal Software Development Engineer · Phoenix, AZ

Agenda

01 **The Edge Observability Problem**
Why retail edge monitoring is uniquely hard

02 **Infrastructure Modernization**
From VMs to lightweight Kubernetes (K3s/MicroK8s)

03 **Prometheus vs. OpenTelemetry**
Why Prometheus fails at the edge — and how OTel replaces it

04 **OTel for Metrics, Traces & Logs**
End-to-end telemetry pipeline design

05 **Frontend Observability**
WebVitals, session traces, UI-to-backend correlation

06 **Results & Lessons Learned**
Real production numbers and takeaways

The Edge Observability Problem

Retail edge environments face unique constraints that traditional monitoring tools weren't built for.

Resource Constraints

CPU, memory & storage limitations on edge devices — no vertical/horizontal scaling available

Network Bandwidth Limits

Monitoring data takes lowest priority; rural locations have as little as 25-50Mbps shared bandwidth

Tooling Complexity

Traditional tools not designed for edge; expensive and architecturally heavy for distributed use

Why It Matters

Poor edge observability means higher MTTD/MTTR, poor user experience, and lost business insight

Infrastructure Modernization

BEFORE — Traditional VMs

- ✕ Physical / Virtual Machines (non-Kubernetes)
- ✕ Requires node exporters as extra components
- ✕ Additional failure points & architectural complexity
- ✕ Hard to scale observability tooling
- ✕ Resource-heavy Prometheus stack (~3.5GB RAM)

Infrastructure Modernization

AFTER — Lightweight Kubernetes

- ✓ K3s / MicroK8s for edge deployments
- ✓ Kubernetes natively exposes monitoring metrics
- ✓ Fewer external dependencies, simpler architecture
- ✓ CNCF-native telemetry (OTel, Fluentbit, Fluentd)
- ✓ OTel optimized for constrained edge resources

Cloud Native Computing Foundation (CNCF) components enable built-in observability with minimal overhead.

Prometheus vs. OpenTelemetry

Why Prometheus reaches its limits at the retail edge

Capability	Prometheus	OpenTelemetry
Metrics	✓ Pull-based, PromQL	✓ Push/pull, vendor-agnostic
Traces	✗ Not supported	✓ Distributed tracing (Tempo)
Logs	✗ Not supported	✓ OTel SDK (Loki/Splunk)

Prometheus vs. OpenTelemetry

Why Prometheus reaches its limits at the retail edge

Capability	Prometheus	OpenTelemetry
Frontend (UI) RUM	✗ Not supported	✓ WebVitals, sessions, AJAX
Memory @ Edge	~3.5GB ❌	~1.3GB ✓ (63% less)
Network Bandwidth	18–25 Mbps ❌	~650 Kbps ✓ (96% less)
Edge Suitability	Limited	Optimized

OTel Metrics: Production Results

One of the first production-scale OTel deployments for retail edge — real numbers from the field

–63%

3.5GB → 1.3GB

Memory

–96%

18Mbps → 650Kbps

Network Bandwidth

~25

vs. hundreds

Essential Metrics Collected

OTel Metrics: Production Results

How We Got There

OTel Operator

Straightforward to deploy; manages collector lifecycle

Metric Compatibility Fix

Known bug with some backends — affected metrics excluded from processing

Data Type Mismatch

Integer/double conflicts resolved by excluding non-critical metrics or adjusting types

Compression + Selective Collection

Only ~25 essential metrics: cluster state (kube-state), pod (cAdvisor), app (Istio), host (node-exporter)

OTel Distributed Tracing

Implementation

- Configure OTel Collector for trace ingestion, processing & forwarding to Tempo backend
- Define backend endpoint, queue size, and security settings in collector config
- Instrument application workloads with OTel libraries at startup
- Enable trace context propagation across service boundaries
- Full visibility into request flows and latency bottlenecks across the stack

OTel Distributed Tracing

Resource Impact

Negligible CPU overhead

Minimal impact on Collector and instrumented services

Minimal bandwidth usage

Efficient trace batching and compression for edge networks

Metrics correlation

Link metrics and service-level interactions end-to-end

Latency root cause

Pinpoint bottlenecks across microservice boundaries

Logging: The Critical Edge Pillar

Application Logging Standards

- Structured JSON format for consistent parsing
- Include timestamp, EXIT codes, ERROR tags
- Minimize debug logs — emit only when needed
- Machine-readable = faster triage + alerting

Logging: The Critical Edge Pillar

Optimization Techniques

Compression at source
Significantly reduces bandwidth utilization

Selective filtering
By namespace, pod labels, severity, message codes

Rate limiting & buffering
Prevent log storms during failures or traffic spikes

60% log volume reduction
While maintaining full diagnostic capability

Frontend Observability

End-to-end visibility: from browser to backend across distributed retail edge

WebVital Metrics (RUM)

- → LCP — Largest Contentful Paint
- → INP — Interaction to Next Paint
- → CLS — Cumulative Layout Shift
- → HTTP/AJAX request spans
- → Resource load & document times

Frontend Observability

End-to-end visibility: from browser to backend across distributed retail edge

Session-Level Logging

- → Forwarded to Splunk via HEC
- → Session IDs, facility codes, POS attributes
- → Trace/span IDs for backend correlation
- → Geo/edge-specific issue isolation
- → Region & device-type filtering

Frontend Observability

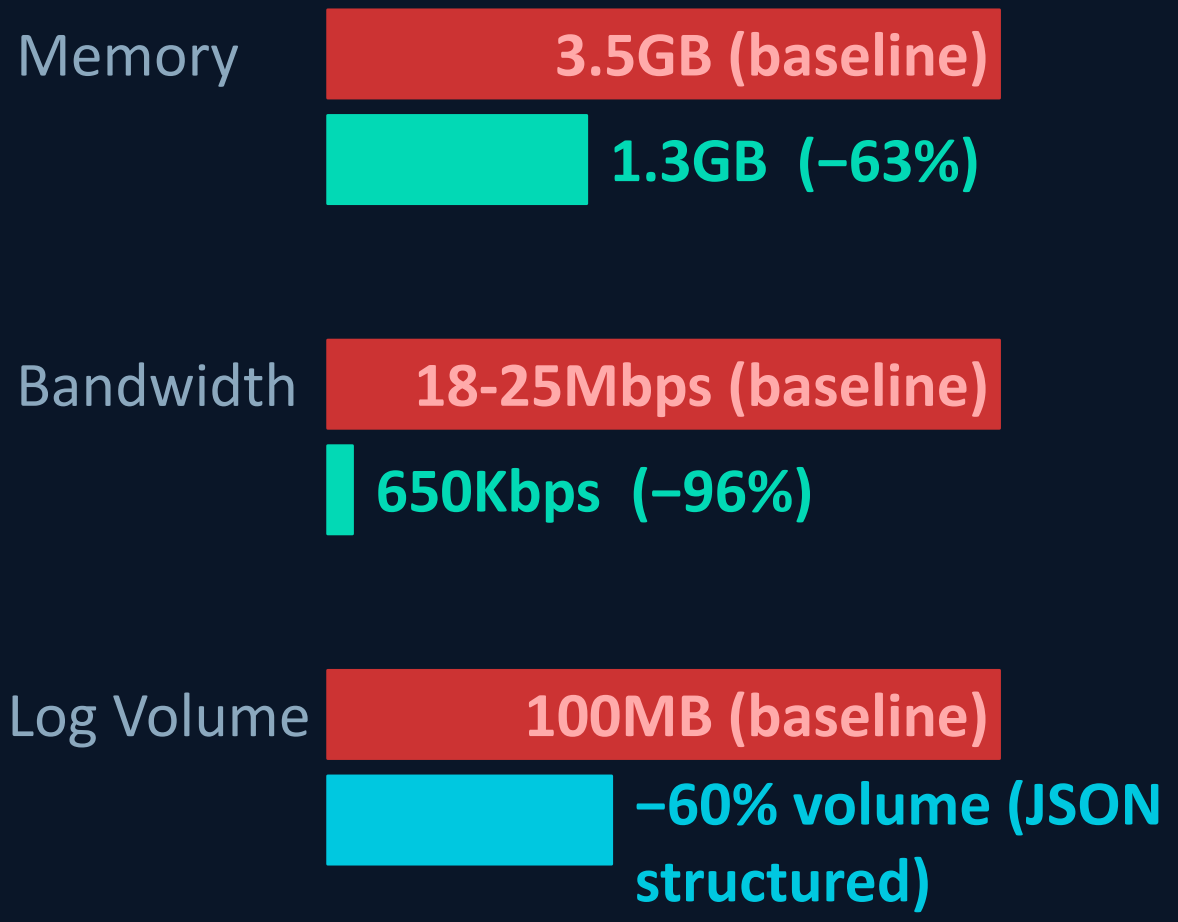
End-to-end visibility: from browser to backend across distributed retail edge

UI→Backend Trace Correlation

- → Shared trace IDs link frontend → backend
- → Transaction-level observability
- → Root cause spans across client + server
- → Funnel & behavior aggregation in dashboards
- → Detect checkout drop-offs in real time

Results & Lessons Learned

Performance Gains vs. Prometheus Baseline



Key Lessons

1. **Bandwidth first**
   More constraining than CPU or memory — design for it upfront
2. **Structured logging pays off**
   JSON + error codes saves hours during incidents
3. **Integration planning**
   UI-to-backend tracing needs upfront architectural decisions
4. **Start with essentials**
   ~25 critical metrics beats collecting everything

Beyond Retail: The Framework Adapts

Edge constraints are universal — the same principles apply across industries

Healthcare

- Remote patient device & vitals monitoring
- Compliance audit trails
- HIPAA-ready security layer for medical data

Telecom / 5G

- Monitor 5G network service delivery
- Track call quality and data routing
- Optimize for variable coverage conditions

Beyond Retail: The Framework Adapts

Edge constraints are universal — the same principles apply across industries

Energy & Utilities

- Smart grid & renewable energy systems
- Track energy production/consumption patterns
- Regulatory compliance & environmental reporting

Future Roadmap

- AI/ML for predictive analytics & auto-detection
- Correlation and auto-resolution
- Security monitoring & privacy protection

Key Takeaways

⚡ 96% bandwidth reduction with OTel vs. Prometheus

🧠 63% memory reduction enables true edge deployment

🔗 Full-stack visibility: browser → backend via shared trace IDs

📋 Structured logs reduce volume 60% with no diagnostic loss

Thank You

Questions & Discussion

Prakash Velusamy

Principal Software Development Engineer ·
Phoenix, AZ

SREcon26 Americas · Seattle, WA · March 24–26, 2026