Prometheus Best Practices — USE vs RED

Prometheus Best Practices and Pitfalls

Source: YouTube Learning — Julius Volz (Prometheus co-founder), Swiss Cloud Native Day 2021
Indexed: 2026-06-15 (MC #103620)

USE vs RED: Decision Framework

USE Method (Resource-Oriented Systems)

For **infrastructure components** (CPU, memory, disk, network):

- **U**tilization: % busy (0-100%)
- **S**aturation: degree of queuing (wait time, queue length)
- **E**rrors: error count/rate

When to use: Cloud Run instances, Azure Container Apps, database connections, worker threads, storage volumes.

RED Method (Request-Oriented Systems)

For **services** handling requests:

- **R**ate: requests/second
- **E**rrors: failed request count or %
- **D**uration: latency (p50, p95, p99)

When to use: REST APIs, BFF layers, RPC services, HTTP endpoints.

Custom Metrics in Application Code

Best Practices

1. **Counter** for events that only go up (requests, errors, jobs completed)
2. **Gauge** for values that go up/down (active connections, queue size, temperature)
3. **Histogram** for bucketed observations (latency, request size) — auto-generates `_sum`, `_count`, `_bucket`
4. **Summary** for client-side quantiles (use histogram + server-side quantiles in PromQL instead)

Common Pitfalls

- **High cardinality labels** (user IDs, UUIDs, timestamps) → cardinality explosion → OOM
- **Missing units in metric names** (`http_request_duration` vs `http_request_duration_seconds`)
- **Inconsistent naming** (mix of snake_case/camelCase)
- **Not exposing `/metrics` endpoint** early in service development
- **Using Summary instead of Histogram** (histograms aggregate better)

PromQL Essentials

```
# Rate of HTTP errors over 5min
rate(http_requests_total{status=~"5.."}[5m])

# 95th percentile latency
histogram_quantile(0.95, rate(http_request_duration_seconds_bucket[5m]))

# CPU utilization (USE)
100 - (avg by (instance) (rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)

# Error rate (RED)
sum(rate(http_requests_total{status=~"5.."}[5m])) / sum(rate(http_requests_total[5m]))
```

How This Applies to ALAI

Current Infrastructure

- **Grafana:** https://grafana.alai.no (monitoring hub)
- **Bilko APIs/BFF:** Java/Spring Boot → **RED metrics** for `/api/*` endpoints
- **LumisCare BFF/services:** Kotlin/Ktor → **RED metrics** for REST + **USE metrics** for connection pools
- **Cloud Run / Azure Container Apps:** Platform exposes **USE metrics** (CPU, memory, request queue)

Recommended Next Steps

1. **Instrument Bilko/LumisCare services** with Micrometer (auto-exposes Prometheus `/actuator/prometheus`)
2. **Add RED dashboards** for all user-facing APIs (Grafana template: https://grafana.com/grafana/dashboards/4701)
3. **Add USE dashboards** for Cloud Run / ACA resource health
4. **Alert on SLIs:** Error rate >1%, p95 latency >2s, CPU >80%

ALAI-Specific Pitfall to Avoid

Do NOT add per-user or per-client labels to core metrics. Use `organization_id` buckets (max ~50) or aggregate at service level. High cardinality = Prometheus death.

References

- Prometheus docs: https://prometheus.io/docs/practices/naming/
- USE Method: http://www.brendangregg.com/usemethod.html
- RED Method: https://grafana.com/blog/2018/08/02/the-red-method-how-to-instrument-your-services/
- Micrometer + Spring Boot: https://micrometer.io/docs/registry/prometheus

---

Revision #1
Created 2026-06-15 14:34:18 UTC by John
Updated 2026-06-15 14:34:18 UTC by John