Style-Guide: Einheitliche Namenskonvention für Metriken bei Signal Iduna

IN REVIEW

Dieses Dokument dient als offizieller Leitfaden für die Benennung von Metriken in allen unseren Systemen. Die Einhaltung dieser Konvention ist entscheidend, um eine konsistente, verständliche und effiziente Observability-Landschaft zu schaffen.

1. Unsere Intention und Philosophie

Jede Metrik muss einfach zu finden, zu verstehen und abzufragen sein, ganz gleich, welcher Service sie erzeugt.

Das Ziel ist es, die kognitive Last für alle Entwickler zu reduzieren und die Zusammenarbeit zwischen den Teams zu fördern. Wenn alle die gleiche technische Sprache sprechen, können wir Probleme schneller diagnostizieren, Muster erkennen und letztendlich bessere Systeme bauen. Wir standardisieren daher auf eine logische, hierarchische Struktur, die von etablierten Standards wie OpenTelemetry inspiriert ist.

---

2. Die Grundstruktur: Logisch und Hierarchisch

Jeder Metrikname folgt einer einfachen, durch Punkte getrennten Struktur. Diese hilft dir, Metriken in Monitoring-Tools wie Grafana oder Prometheus intuitiv zu finden, ähnlich wie du durch ein Dateisystem navigierst.

`bereich.komponente.messwert`

Bereich (Optional, aber stark empfohlen)

Dies ist die logische Domäne der Metrik, die den groben Kontext vorgibt.

> ***Intention:*** *Durch die Einteilung in Bereiche wie `http` oder db weißt du sofort, wo du suchen musst. Dies ermöglicht es dir auch, ganze Klassen von Metriken zu aggregieren (z. B. die Latenz aller Datenbank-Interaktionen), selbst wenn sie von unterschiedlichen Services kommen.*

Bereich	Beschreibung
app	**Der wichtigste Bereich für uns!** Hier gehört deine spezifische Business-Logik hinein.
`http`	Für alles, was mit HTTP-Servern oder -Clients zu tun hat.
`db`	Für Datenbank-Clients und -Verbindungen.
`rpc`	Für gRPC oder andere Remote Procedure Calls.
`jvm`	Für interne Metriken der Java Virtual Machine (Garbage Collection, Memory, Threads etc.).
`kafka`	Für Kafka-Producer oder -Consumer.

Komponente

Das spezifische System oder die logische Einheit, die du misst.

> ***Intention:*** *Dieser Teil benennt das "Was". Er schafft Klarheit darüber, welche Komponente innerhalb eines Bereichs instrumentiert wird. Ein spezifischer Name wie `ktg_antrag` sagt sofort mehr aus als ein generischer Begriff.*

Beispiele
`server`, `client`
`connections`, `requests`
`ktg_antrag`, `rohdaten_prozessor` (Beispiele für unsere spezifischen Service-Komponenten)

Messwert

Was genau wird gemessen? Dies sollte immer eine **Einheit im Singular** und in **Basiseinheiten** (z.B. `seconds`, `bytes`) sein.

> ***Intention:*** *Einheitlichkeit ist hier absolut entscheidend. Wenn `duration` immer in Sekunden und `size` immer in Bytes angegeben wird, vermeidest du gefährliche Fehlinterpretationen und umständliche Umrechnungen in deinen Abfragen. Singularformen halten die Namen zudem kurz und prägnant.*

Messwert	Beschreibung	Einheit
`duration`	Dauer einer Operation	Sekunden (seconds)
`size`	Größe von Daten	Bytes
`count`	Eine generische Anzahl	(Einheitslos)
`errors`	Anzahl von Fehlern	(Einheitslos)
`lag`	Verzögerung in einer Queue oder einem Stream	(Einheitslos, z.B. Anzahl Nachrichten)
`usage`	Auslastung als Verhältnis (Ratio)	0 bis 1

---

3. Standardisierte Suffixe für Metrik-Typen

Um den Typ einer Metrik auf den ersten Blick zu erkennen und korrekt abzufragen, verwenden wir standardisierte Suffixe.

`_total` für Counter

Dies ist ein kumulativer Wert, der nur ansteigen kann (z.B. die Gesamtzahl der verarbeiteten Anfragen seit dem letzten Neustart).

> ***Intention:*** *Das Suffix `_total` signalisiert deinem Monitoring-Tool (z.B. Prometheus), dass du hier Aggregationsfunktionen wie `rate()` oder `increase()` anwenden kannst, um Raten zu berechnen (z.B. Anfragen pro Sekunde). Bei anderen Metrik-Typen wäre dies nicht sinnvoll.*
>
> ***Beispiel:*** *`http.server.requests.total`*

`_bucket` / `_sum` / `_count` für Histogramme

Histogramme werden zur Berechnung von Quantilen (p95, p99) für Latenzen oder Größenverteilungen verwendet.

> ***Intention:*** *Statt nur einen Durchschnittswert zu speichern, der durch Ausreißer stark verzerrt sein kann, gibt dir ein Histogramm ein vollständiges Bild der Verteilung. Du kannst damit präzise Service Level Objectives (SLOs) definieren, wie z.B. "99% aller Anfragen müssen unter 200ms beantwortet werden".*
>
> ***Beispiel:*** *`http.server.request.duration_bucket` (wird immer von `..._sum` und `..._count` begleitet)*

(Kein Suffix) für Gauges

Dies ist ein Wert, der sowohl steigen als auch fallen kann (z.B. die aktuell aktive Anzahl von Verbindungen oder die aktuelle CPU-Last).

> ***Intention:*** *Die Abwesenheit eines Suffixes ist ein bewusstes Signal. Du weißt sofort, dass dieser Wert einen aktuellen Zustand repräsentiert und nicht über die Zeit aggregiert werden sollte, um eine Rate zu bilden.*
>
> ***Beispiel:*** *`jvm.threads.live`*

---

4. Transformation: Vorher vs. Nachher

Hier siehst du, wie deine bisherigen, oft uneinheitlichen Metriken in die neue, saubere Welt übersetzt werden. Dies verdeutlicht den Mehrwert der neuen Struktur.

Kategorie	Alt (Beispiele deiner Metriken)	Neu (Unsere Konvention)	Anmerkung (Die Intention dahinter)
Fehler	`BRErrorCountAll`, `RohdatenErrorCount`, `http_responseCodes_serverError_total`	`http.server.requests.errors.total` (mit Label `http.route` oder `app.name`)	**Intention:** Statt vieler, schwer zu findender Fehlermetriken gibt es nur noch eine. Den Kontext (welcher Service oder Endpunkt) liefert ein Label. Das macht es trivial, die Fehlerrate über das gesamte System zu aggregieren.
Latenz	`imsKvlAbrechnungLesenService_sum`, `Tomcat_GlobalRequestProcessor_maxTime`, `Confluence_RequestMetrics_AverageExecutionTimeForLastTenRequests`	`app.ims.abrechnung.duration` (als Histogramm)	**Intention:** Ein einziges Histogramm ist unendlich mächtiger als vorab berechnete Summen oder Durchschnitte. Daraus kannst du *alles* berechnen (p95, Median, Durchschnitt) und zwar mit hoher Genauigkeit.
Anzahl	`BRRequestCountAll`, `RohdatenCount`, `kafka_messages_processed_successfully_total`	`http.server.requests.total`, `app.rohdaten.verarbeitet.total`, `kafka.consumer.messages.processed.total`	**Intention:** Die klaren Präfixe (`http`, `app`, `kafka`) sorgen für Ordnung. Das `_total`-Suffix stellt sicher, dass jeder weiß, dass es sich um einen Counter für Ratenberechnungen handelt.
Sättigung	`hikaricp_connections_pending`, `Confluence_MailTaskQueue_TasksSize`	`db.client.connections.pending`, `app.confluence.mail.queue.size`, `kafka.consumer.lag`	**Intention:** Die Namen sind kürzer, prägnanter und folgen der Bereichs-Logik. Dies sind Gauges (kein Suffix), die einen aktuellen Füllstand anzeigen.
Business	`signaliduna_ktg_aktion_ADVNummerAbfrageErfolgCounter`	`app.ktg.adv_abfrage.total{ergebnis="erfolg"}`	**Intention:** Dies ist der mächtigste Wandel! Statt für jeden Ausgang (Erfolg, Fehler) eine eigene Metrik anzulegen, verwenden wir **eine Metrik mit einem Ergebnis-Label**. Damit kannst du flexible Abfragen bauen, z.B. das Verhältnis von Erfolg zu Fehler berechnen.

5. Do's & Don'ts: Die goldenen Regeln

Do's (Das solltest du tun)	Don'ts (Das solltest du vermeiden)	Intention
Verwende `.` als Trennzeichen.	Verwende _ oder **`camelCase`**im Namen.	Inkonsistenz erschwert die automatische Vervollständigung in Abfrage-Editoren.
Verwende den **`app.`**-Präfix für anwendungsspezifische Business-Metriken.	Vermische Business-Metriken mit Infrastruktur-Metriken.	Eine klare Trennung hilft, den Überblick zu behalten und Business-KPIs von System-Metriken zu unterscheiden.
Verwende **Labels**, um Dimensionen zu trennen (`...total{status="ok"}`).	Kodiere den Wert in den Metriknamen (`...ok_total`, `...error_total`).	Das führt zu einer "Explosion der Metriken" (hohe Kardinalität) und macht Aggregationen unnötig kompliziert.
Verwende **Basiseinheiten**: `seconds`, `bytes`, `bits`.	Verwende gemischte Einheiten wie `millisecondsohne` Kennzeichnung.	Dies ist eine häufige und gefährliche Fehlerquelle. Basiseinheiten schaffen eine verlässliche Grundlage.
Halte Namen **kurz, aber verständlich**.	Verwende Abkürzungen, die nur der ursprüngliche Autor versteht.	`db.client.connections.usage` ist für jeden verständlich, `db.cl.con.usg` nicht.
Dokumentiere die wichtigsten Business-Metriken an einem zentralen Ort (z.B. Confluence).	Gehe davon aus, dass jeder die Bedeutung von `TIE-MK-Test._S6` kennt.	Gute Metriken sind selbsterklärend, aber komplexe Business-Metriken brauchen Kontext.
Sei konsistent. Die wichtigste Regel von allen.	Erlaube jedem Team, eigene Konventionen zu erfinden.	Konsistenz übertrifft Perfektion. Eine einheitliche Regel ist besser als das Chaos vieler verschiedener Regeln.

6. Migrationsstrategie: Der Weg in die neue Welt

Eine Umstellung über Nacht ist unrealistisch und riskant. Verfolge stattdessen diesen pragmatischen Ansatz, um den Übergang reibungslos zu gestalten.

Warum ist die Migration notwendig?

Die aktuelle Vielfalt an Namensschemata führt zu Ineffizienz, Fehlern und einer hohen Einstiegshürde für neue Teammitglieder. Dashboards sind schwer zu warten, Alarme sind inkonsistent und die teamübergreifende Analyse von Problemen wird zum Ratespiel. Durch die Vereinheitlichung **reduzieren wir langfristig Kosten, senken die Komplexität und erhöhen die Geschwindigkeit**, mit der wir auf Vorfälle reagieren können.

Der Migrationspfad

Schritt	Aktion	Intention
1. Dokumentieren & Kommunizieren	Mache diesen Guide offiziell und für alle Entwickler leicht zugänglich (z.B. im Confluence). Führe Schulungen durch.	Ein gemeinsames Verständnis und die offizielle Verankerung sind der erste und wichtigste Schritt zur Akzeptanz.
2. Dual-Writing implementieren	Wenn du einen Service auf OpenTelemetry umstellst, lass ihn für eine Übergangszeit (z.B. 4 Wochen) **sowohl die alten als auch die neuen Metriken** senden.	Dies ist dein Sicherheitsnetz. Es stellt sicher, dass du keine Lücken in deiner Überwachung hast, während du die neuen Dashboards und Alarme aufbaust und validierst.
3. Dashboards & Alarme migrieren	Baue deine wichtigsten Dashboards und Alarme parallel mit den neuen, sauberen Metriknamen nach.	So kannst du direkt vergleichen, ob die neuen Metriken die gleichen Informationen liefern. Dies schafft Vertrauen in das neue System, bevor du das alte abschaltest.
4. Alte Metriken abschalten	Sobald die Migration validiert ist, entferne die alte Instrumentierung oder filtere die alten Metriken auf Collector-Ebene (z.B. in Cribl oder dem OpenTelemetry Collector) heraus.	Das ist der Moment, in dem du die Früchte deiner Arbeit erntest: **dauerhaft reduzierte Kosten** (weniger Daten) und **geringere Komplexität** (weniger Chaos).