

Sortierung und Einordnung der Dimensionen

- Sortierung und Einordnung der Dimensionen
 - 1. Absolute Kern-Identifikatoren
 - 2. Kontext der Laufzeitumgebung (Kubernetes & Host)
 - 3. Kontext von Anfragen und Kommunikation
 - 4. Spezifischer Technologie- & Applikationskontext
 - 5. Telemetrie- und SDK-Kontext
 - 6. Generische & weniger wichtige Dimensionen
 - 7. Eindeutige und hochdynamische IDs (Hohe Kardinalität!)
 - 8. Unbegrenzte oder variable Textinhalte (Hohe Kardinalität!)
 - 9. Redundante oder zu granulare Dimensionen
- Beantwortung deiner Fragen
 - Welche Dimensionen sollten entfernt werden, weil sie Einfluss auf die Kardinalität haben?
 - Welche Dimensionen sollten aus Datenschutzgründen entfernt werden?
 - Wie sieht ein Migrationsplan aus zu weniger Metriken mit mehr Aussagekraft?

Sortierung und Einordnung der Dimensionen

Hier ist die Einordnung deiner Dimensionen nach den von dir vorgegebenen Gruppen, sortiert nach absteigender Wichtigkeit innerhalb jeder Gruppe. Die Wichtigkeit beurteile ich danach, wie gut eine Dimension zur Korrelation und zum Filtern von Problemen über verschiedene Systemgrenzen hinweg beiträgt.

1. Absolute Kern-Identifikatoren

Dies sind die wichtigsten Dimensionen. Sie definieren den "Wo"-Kontext einer Metrik und sind für das Slicing und Dicing von Daten unerlässlich. Sie sollten in *jeder* Metrik vorhanden sein, wie es auch dein ADR ([COPLA-0009-Labeling-Konzept](#)) fordert.

Dimension	Wichtigkeit & Begründung
<code>service.name</code>	Extrem Hoch: Das ist die grundlegendste Information. Ohne den Service-Namen ist eine Metrik fast wertlos.
<code>deployment.environment</code>	Extrem Hoch: Kritisch, um zwischen <code>prod</code> , <code>dev</code> , <code>staging</code> etc. zu unterscheiden. Grundlage für Alarme.
<code>k8s.cluster.name</code> / <code>si.clustername</code>	Extrem Hoch: Identifiziert den Kubernetes-Cluster. Wichtig in Multi-Cluster-Umgebungen. Es scheint, ihr habt hier zwei redundante Labels, die konsolidiert werden sollten.
<code>si.pipeline</code>	Hoch: Scheint ein für euch wichtiger, interner Kontext zu sein (laut ADR). Essentiell für das Verständnis von Datenflüssen.
<code>ciid</code>	Hoch: Ebenfalls ein zentraler, interner Identifikator laut ADR.
<code>service.version</code>	Hoch: Ermöglicht die Analyse von Problemen im Kontext von neuen Deployments (z.B. "Performance-Problem seit Version X").

2. Kontext der Laufzeitumgebung (Kubernetes & Host)

Diese Dimensionen geben detaillierten Aufschluss über die Infrastruktur, auf der dein Service läuft. Sie sind sehr nützlich für das Debugging von Infrastruktur-nahen Problemen.

Dimension	Wichtigkeit & Begründung
<code>k8s.namespace.name</code>	Hoch: Trennt verschiedene Applikationen und Teams innerhalb eines Clusters.
<code>host.name</code> / <code>k8s.node.name</code>	Mittel: Nützlich, um Probleme zu identifizieren, die auf einen bestimmten Host oder Node beschränkt sind. Könnte redundant sein.
<code>k8s.pod.name</code>	Mittel bis Niedrig: Hilfreich für Ad-hoc-Analysen, aber Vorsicht vor hoher Kardinalität bei häufigen Deployments.
<code>os.type</code> / <code>host.arch</code>	Niedrig: Meist statisch. Nur relevant, wenn ihr heterogene Betriebssysteme oder Architekturen betreibt.
<code>container.image.name</code> / <code>container.image.tag</code>	Niedrig: Oft schon durch <code>service.name</code> und <code>service.version</code> abgedeckt. Kann nützlich sein, ist aber oft redundant.

3. Kontext von Anfragen und Kommunikation

Diese Dimensionen sind entscheidend für das Verständnis von Service-Interaktionen und externer Kommunikation.

Dimension	Wichtigkeit & Begründung
<code>http.request.method / sf_httpMethod</code>	Hoch: Grundlegende Unterscheidung von Requests (GET, POST etc.). Niedrige Kardinalität.
<code>http.response.status_code / status_code</code>	Hoch: Essentiell, um Fehlerraten (4xx, 5xx) zu überwachen.
<code>url.scheme / http.scheme</code>	Mittel: Unterscheidung zwischen HTTP und HTTPS. Normalerweise niedrige Kardinalität.
<code>server.address / net.host.name</code>	Mittel: Kann zur Identifizierung von Zielsystemen verwendet werden. Kann aber auch zu hoher Kardinalität führen.
<code>server.port / net.host.port</code>	Mittel: Wichtig, wenn Services auf mehreren Ports lauschen.

4. Spezifischer Technologie- & Applikationskontext

Diese Labels sind Gold wert, wenn sie spezifisch für eine Technologie sind, aber sie sollten nur dort verwendet werden, wo sie auch relevant sind.

Dimension	Wichtigkeit & Begründung
<code>database / dbname</code>	Hoch: Unverzichtbar für Metriken, die Datenbank-Interaktionen messen.
<code>topic</code>	Hoch: Für Kafka-Metriken absolut entscheidend.
<code>cache / cache.manager</code>	Mittel: Nützlich zur Analyse von Caching-Performance und -Problemen.
<code>spring.security.*, jvm.*, kafka_version</code>	Niedrig bis Mittel: Sehr spezifisch. Wertvoll für tiefgehende Analysen der jeweiligen Technologie, aber nicht für allgemeine Metriken.

5. Telemetrie- und SDK-Kontext

Diese sind meist für die Fehlersuche im Telemetrie-System selbst nützlich.

Dimension	Wichtigkeit & Begründung
<code>telemetry.sdk.name</code>	Niedrig: Gut zu wissen, welches SDK die Daten sendet.
<code>telemetry.sdk.language</code>	Niedrig: Hilft bei der Zuordnung zu Teams oder Technologien.
<code>telemetry.sdk.version</code>	Niedrig: Wichtig bei Problemen mit dem SDK selbst (z.B. fehlerhafte Metriken nach einem Update).

6. Generische & weniger wichtige Dimensionen

Hier landen Labels, die oft zu unspezifisch sind oder deren Nutzen begrenzt ist.

Dimension	Wichtigkeit & Begründung
<code>status, type, name, outcome, result</code>	Sehr niedrig: Diese Labels sind extrem generisch. Ohne den Kontext des Metriknamens sind sie kaum aussagekräftig. <code>status</code> könnte z.B. "success" oder "error" bedeuten, was besser in einem Label wie <code>http.response.status_code</code> aufgehoben wäre. Sollten oft entfernt und durch spezifischere Labels ersetzt werden.
<code>os.description</code>	Sehr niedrig: Zu detailliert und ändert sich selten. Besser in Logs oder als Host-Eigenschaft aufgehoben.

7. Eindeutige und hochdynamische IDs (Hohe Kardinalität!)

WARNUNG: Diese Gruppe ist der Hauptverursacher für explodierende Kosten und Performance-Probleme. Jede Dimension hier ist ein Kandidat für die Entfernung.

Dimension	Wichtigkeit & Begründung
<code>k8s.pod.uid</code>	Kandidat zum Entfernen: Eindeutig pro Pod-Lebenszyklus. Jedes neue Deployment erzeugt neue Werte. Extrem hohe Kardinalität.
<code>service.instance.id</code>	Kandidat zum Entfernen: Eindeutig pro Instanz. Führt zu einer separaten Zeitreihe für jeden Pod/jede Instanz.
<code>process.pid</code>	Kandidat zum Entfernen: Eindeutig pro Prozess. Startet eine Applikation neu, entsteht eine neue Zeitreihe.
<code>container.id</code>	Kandidat zum Entfernen: Eindeutig pro Container. Sehr hohe Kardinalität.

tokenId, client_id, client-id, clientId	Kandidat zum Entfernen/Prüfen: Wenn dies eindeutige Session- oder Request-IDs sind, ist die Kardinalität unbegrenzt. Rote Flagge!
---	---

8. Unbegrenzte oder variable Textinhalte (Hohe Kardinalität!)

WARNING: Ähnlich wie die vorige Gruppe, aber hier handelt es sich um variable Strings statt IDs.

Dimension	Wichtigkeit & Begründung
error, exception, class, cause	Kandidat zum Entfernen: Fehlermeldungen oder Klassennamen gehören in Logs , nicht in Metrik-Dimensionen. Die Anzahl der möglichen Werte ist praktisch unbegrenzt.
uri, http.route, path	Kandidat zum Entfernen: Wenn die URIs IDs enthalten (z.B. /users/12345/profile), führt dies zu unendlicher Kardinalität. Wenn, dann nur mit parametrisierten Routen wie /users/{userId}/profile verwenden.
process.command_line	Kandidat zum Entfernen: Kann sehr lang und variabel sein. Gehört in Host-Metadaten oder Logs.

9. Redundante oder zu granulare Dimensionen

Diese Dimensionen bieten oft Informationen, die bereits in anderen, besseren Dimensionen vorhanden sind.

Dimension	Wichtigkeit & Begründung
k8s.namespace.labels.si.pipeline	Redundant: Der Wert sollte bereits im Label si.pipeline vorhanden sein, wie vom ADR gefordert.
k8s.namespace.labels.ciid	Redundant: Dasselbe gilt hier; der Wert sollte im Label ciid stehen.
si.environment, sf_environment	Redundant: Scheint dasselbe wie deployment.environment zu sein. Sollte konsolidiert werden.
si.service, sf_service, service_name	Redundant: Scheint dasselbe wie service.name zu sein. Konsolidieren!

Beantwortung deiner Fragen

Welche Dimensionen sollten entfernt werden, weil sie Einfluss auf die Kardinalität haben?

Du solltest dringend alle Dimensionen aus den Gruppen "7. Eindeutige und hochdynamische IDs" und "8. Unbegrenzte oder variable Textinhalte" überprüfen und in den meisten Fällen entfernen.

Top-Kandidaten zum sofortigen Entfernen:

- k8s.pod.uid, service.instance.id, process.pid, container.id: Diese identifizieren eine spezifische, kurzlebige Instanz. Aggregierte Metriken sollten über service.name, deployment.environment etc. gebildet werden. Für die Analyse einer einzelnen Instanz sind Traces und Logs das richtige Werkzeug.
- error, exception, uri (wenn nicht parametrisiert), process.command_line: Diese haben eine unbegrenzte Anzahl möglicher Werte. Jeder einzigartige Wert erzeugt eine neue Zeitreihe.

Warum? Die Kardinalität ist die Anzahl der einzigartigen Zeitreihen, die durch die Kombination aller Dimensionswerte entstehen. Wenn du eine Dimension mit 10.000 einzigartigen Werten (z.B. service.instance.id) hast, erzeugst du 10.000 Zeitreihen *pro Metrik*. Das macht Abfragen langsam, teuer und oft unmöglich.

Welche Dimensionen sollten aus Datenschutzgründen entfernt werden?

Dies erfordert eine genaue Prüfung, aber hier sind die verdächtigsten Kandidaten:

- client_id, clientId, client-id, username, user: Diese könnten eine direkte ID eines Benutzers enthalten, was ein klares Datenschutzproblem darstellt (PII - Personally Identifiable Information).
- net.host.name, server.address, hostname: Könnten potenziell IP-Adressen oder benutzerspezifische Hostnamen enthalten, was ebenfalls als PII gewertet werden kann.
- process.command_line: Könnte versehentlich sensible Argumente wie Passwörter oder Tokens enthalten.

Empfehlung: Setze dich mit deinem Datenschutzbeauftragten zusammen und überprüfe diese Dimensionen. Im Zweifel sollten sie entfernt oder die Werte anonymisiert/gehasht werden (wobei Hashing die Kardinalität nicht reduziert).

Wie sieht ein Migrationsplan aus zu weniger Metriken mit mehr Aussagekraft?

Ein solcher Plan sollte schrittweise erfolgen, um den Betrieb nicht zu stören.

Phase 1: Analyse & Definition (1-2 Wochen)

1. **Audit durchführen:** Identifiziere, welche Teams und Dashboards die problematischen, hochkardinalen Dimensionen (siehe oben) verwenden. Finde heraus, *welche Fragen* sie damit beantworten wollen.

2. **"Golden Signals" definieren:** Definiert pro Service-Typ (z.B. Web-API, Batch-Job) einen minimalen Satz an Kernmetriken (die "Golden Signals": Latenz, Traffic, Fehler, Sättigung).
3. **"Golden Labels" festlegen:** Finalisiert die Liste der absolut notwendigen, standardisierten Labels. Euer ADR ist hierfür die perfekte Grundlage. Das sollten die **"Absoluten Kern-Identifikatoren"** sein, plus wenige, ausgewählte Kontextvariable.

Phase 2: Konsolidierung & Bereinigung (2-4 Wochen)

1. **Redundanz eliminieren:** Schafft redundante Labels wie `si.clustername` vs. `k8s.cluster.name` ab. Entscheidet euch für *einen* Standard und setzt diesen technisch durch (z.B. im OpenTelemetry Collector).
2. **Hochkardinale Labels ersetzen:**
 - Für Debugging-IDs (`service.instance.id`, `k8s.pod.uid`): Leitet die Teams an, stattdessen **Tracing** zu verwenden. Ein Trace verbindet Logs und Metriken für einen einzelnen Request und ist das perfekte Werkzeug für diese Art von Detailanalyse.
 - Für Fehlertexte (`error`, `exception`): Nutzt **Structured Logging**. Die Metrik kann die *Anzahl* der Fehler mit einem `error_type` (z.B. "database_error", "validation_error") zählen, aber die detaillierte Fehlermeldung gehört in einen Log-Eintrag, der über eine Trace-ID mit dem Request verknüpft ist.
3. **Parametrisierung durchsetzen:** Stellt sicher, dass Routen wie `/users/{userId}` anstelle von `/users/123` als `http.route`-Dimension verwendet werden. Dies muss in der Anwendungsinstrumentierung konfiguriert werden.

Phase 3: Umsetzung & Kommunikation (laufend)

1. **Technische Durchsetzung:** Implementiert die Label-Bereinigung zentral, wenn möglich (z.B. mit einem Processor im OpenTelemetry Collector), um sicherzustellen, dass unerwünschte Labels gar nicht erst im Backend ankommen.
2. **Dokumentation & Schulung:** Dokumentiert die neuen "Golden Labels" und die Best Practices (z.B. "Keine IDs in Metriken!"). Macht Schulungen für die Entwicklungsteams.
3. **Deprecation-Phase:** Kündigt an, dass die alten, unerwünschten Dimensionen in X Wochen/Monaten nicht mehr verfügbar sein werden. Baut Dashboards, die die Nutzung dieser Labels zeigen, um den Fortschritt zu verfolgen.