Workgroup: Network Working Group
Internet-Draft: draft-kll-yang-label-tsdb-00
Published: 18 October 2023
Intended Status: Standards Track
Expires: 20 April 2024
Authors: K. Larsson
Deutsche Telekom

Mapping YANG Data to Label-Set Time Series

Abstract

This document proposes a standardized approach for representing YANG-modeled configuration and state data, for storage in Time Series Databases (TSDBs) that identify time series using a label-set. It outlines procedures for translating YANG data representations to fit within the label-centric structures of TSDBs and vice versa. This mapping ensures clear and efficient storage and querying of YANG-modeled data in TSDBs.

Discussion Venues

This note is to be removed before publishing as an RFC.

Source for this draft and an issue tracker can be found at https://github.com/plajjan/draft-kll-yang-label-tsdb.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 20 April 2024.

Table of Contents

1. Introduction

The aim of this document is to define rules for representing configuration and state data defined using the YANG data modeling language [RFC7950] as time series using a label-centric model.

The majority of modern Time Series Databases (TSDBs) employ a label-centric model. In this structure, time series are identified by a set of labels, each consisting of a key-value pair. These labels facilitate efficient querying, aggregation, and filtering of data over time intervals. Such a model contrasts with the hierarchical nature of YANG-modeled data. The challenge, therefore, lies in ensuring that YANG-defined data, with its inherent structure and depth, can be seamlessly integrated into the flat, label-based structure of most contemporary TSDBs.

This document seeks to bridge this structural gap, laying out rules and guidelines to ensure that YANG-modeled configuration and state data can be effectively stored, queried, and analyzed within label-centric TSDBs.

2. Specification of the Mapping Procedure

Instances of YANG data nodes are mapped to metrics. Only nodes that carry a value are mapped. This includes leafs and presence containers. The hierarchical path to a value, including non-presence containers and lists, form the path that is used as the name of the metric. The path is formed by joining YANG data nodes using _. Special symbols, e.g. -, in node names are replaced with _.

List keys are mapped into labels. The path to the list key is transformed in the same way as the primary name of the metric. Compound keys have each key part as a separate label.

2.1. Example: Packet Counters in IETF Interfaces Model

Consider the in-unicast-pkts leaf from the IETF interfaces model that captures the number of incoming unicast packets on an interface:

Original YANG Instance-Identifier: yang /interfaces/ interface[name='eth0']/statistics/in-unicast-pkts

Following the mapping rules defined:

1. The path components, including containers and list names, are transformed into the metric name by joining the node names with _. Special symbols, e.g. - are replaced with _.

Resulting Metric Name: interfaces_interface_statistics_in_unicast_pkts

1. The list key "predicate", which in this case is the interface name (eth0), is extracted and stored as a separate label. The label key represents the complete path to the key.

Resulting Label: interfaces_interface_name = eth0

1. The leaf value, which represents the actual packet counter, remains unchanged and is directly mapped to the value in the time series database.

For instance, if the packet counter reads 5,432,100 packets:

Value: 5432100

1. As part of the standard labels, a server identification string is also included. A typical choice of identifier might be the hostname. For this example, let's assume the device name is router-01:

Label: host = router-01

Final Mapping in the TSDB:

*Metric: interfaces_interface_statistics_in_unicast_pkts

*Value: 5432100

*Labels:

-host = router-01

-interfaces_interface_name = eth0

2.2. Mapping values

Leaf values are mapped based on their intrinsic type:

*All integer types are mapped to integers and retain their native representation

-some implementations only support floats for numeric values

*decimal64 values are mapped to floats and the value should be rounded and truncated as to minimize the loss of information

*Enumeration types are mapped using their string representation.

*String types remain unchanged.

2.3. Choice

Choice constructs from YANG are disregarded and not enforced during the mapping process. Given the temporal nature of TSDBs, where data spans across time, different choice branches could be active in a single data set, rendering validation and storage restrictions impractical.

2.4. Host / device name

There is an implicit host label identifying the server, typically set to the name of the host originating the time series data.

Instance data retrieved from YANG-based servers do not generally identify the server it originates from. As a time series database is likely going to contain data from multiple servers, the host label is used to identify the source of the data.

3. Querying YANG modeled time series data

The process of storing YANG-modeled data in label-centric TSDBs, as defined in the previous sections, inherently structures the data in a way that leverages the querying capabilities of modern TSDBs. This chapter provides guidelines on how to construct queries to retrieve this data effectively.

3.1. 1. Basic Queries

To retrieve all data points related to incoming unicast packets from the IETF interfaces model:

* **InfluxQL**: sql SELECT * FROM interfaces_interface_statistics_in_unicast_pkts

* **PromQL**: promql interfaces_interface_statistics_in_unicast_pkts

3.2. 2. Filtering by Labels

To retrieve incoming unicast packets specifically for the interface eth0:

* **InfluxQL**: sql SELECT * FROM interfaces_interface_statistics_in_unicast_pkts WHERE interfaces_interface_name = 'eth0'

* **PromQL**: promql interfaces_interface_statistics_in_unicast_pkts{interfaces_interface_name="eth0"}

Similarly, to filter by device / host name:

* **InfluxQL**: sql SELECT * FROM interfaces_interface_statistics_in_unicast_pkts WHERE host = 'router-01'

* **PromQL**: promql interfaces_interface_statistics_in_unicast_pkts{host="router-01"}

3.3. 3. Time-based Queries

* **InfluxQL**: sql SELECT * FROM interfaces_interface_statistics_in_unicast_pkts WHERE time > now() - 24h

Prometheus fetches data based on the configured scrape interval and retention policies, so time-based filters in PromQL often center around the range vectors. For data over the last 24 hours:

* **PromQL**: promql
  interfaces_interface_statistics_in_unicast_pkts[24h]

3.4. 4. Aggregations

To get the average number of incoming unicast packets over the last hour:

* **InfluxQL**: sql SELECT MEAN(value) FROM
  interfaces_interface_statistics_in_unicast_pkts WHERE time > now()
  - 1h GROUP BY time(10m)

* **PromQL**: promql
  avg_over_time(interfaces_interface_statistics_in_unicast_pkts[1h])

3.5. 5. Combining Filters

To retrieve the sum of incoming unicast packets for eth0 on router-01 over the last day:

* **InfluxQL**: sql SELECT SUM(value) FROM
  interfaces_interface_statistics_in_unicast_pkts WHERE
  interfaces_interface_name = 'eth0' AND host = 'router-01' AND time
  > now() - 24h

* **PromQL**: promql
  sum(interfaces_interface_statistics_in_unicast_pkts{interfaces_int
  erface_name="eth0", host="router-01"})[24h]

3.6. 6. Querying Enumeration Types

In YANG models, enumerations are defined types with a set of named values. The oper-status leaf in the IETF interfaces model is an example of such an enumeration, representing the operational status of an interface.

For instance, the oper-status might have values such as up, down, or testing.

To query interfaces that have an oper-status of up:

* **InfluxQL**: sql SELECT * FROM interfaces_interface_oper_status WHERE
  value = 'up'

* **PromQL**: promql interfaces_interface_oper_status{value="up"}

Similarly, to filter interfaces with oper-status of down:

- **InfluxQL**: sql SELECT * FROM interfaces_interface_oper_status WHERE value = 'down'

- **PromQL**: promql interfaces_interface_oper_status{value="down"}

This approach allows us to effectively query interfaces based on their operational status, leveraging the enumeration mapping within the TSDB.

4. Requirements on time series databases

This document specifies a mapping to a conceptual representation, not a particular concrete interface. To effectively support the mapping of YANG-modeled data into a label-centric model, certain requirements must be met by the Time Series Databases (TSDBs). These requirements ensure that the data is stored and retrieved in a consistent and efficient manner.

4.1. Support for String Values

Several YANG leaf types carry string values, including the string type itself and all its descendants as well as enumerations which are saved using their string representation.

The chosen TSDB must support the storage and querying of string values. Not all TSDBs inherently offer this capability, and thus, it's imperative to ensure compatibility.

4.2. Sufficient Path Length

YANG data nodes, especially when representing deep hierarchical structures, can result in long paths. When transformed into metric names or labels within the TSDB, these paths might exceed typical character limits imposed by some databases. It's essential for the TSDB to accommodate these potentially long names to ensure data fidelity and avoid truncation or loss of information.

4.3. High Cardinality

Given the possibility of numerous unique label combinations (especially with dynamic values like interface names, device names, etc.), the chosen TSDB should handle high cardinality efficiently. High cardinality can impact database performance and query times, so it's essential for the TSDB to have mechanisms to manage this efficiently.

5. Normative References

[RFC7950]
Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

Author's Address

Kristian Larsson
Deutsche Telekom

Email: kristian@spritelink.net