Proposal: (Remote Write v1.1) Add metadata as a TimeSeries field in remote write

Author: Paschalis Tsilias (paschalis.tsilias@grafana.com)
Date: Jun 15, 2023
Status: Draft
Stakeholders: Callum Styan, Bartek Płotka
Visibility: Document is public

Background - Prior Art

- Prometheus Remote Write Specification v0.1
- https://prometheus.io/docs/concepts/remote_write_spec/
- Proposal: Staged propagation of metadata for Prometheus Remote Write targets
- 2020-02 Propagate metric metadata via remote write
- https://github.com/prometheus/prometheus/pull/7771
- https://groups.google.com/g/prometheus-developers/c/w-eotGenBGg/m/UZWdxzcBBwAJ

Problem Statement

The current approach for propagating metadata was designed in 2020-02 Propagate metric metadata via remote write and (in a high-level) consists of pulling metadata from the scrape cache on a regular, configurable interval (every 1 minute by default). The metadata is stored per “metric family” and is deduplicated within different targets (first one wins). Then, it is sent as a standalone remote write request, batched with up to N metadata entries (defaults to 500). In case of a horizontally-scaled Prometheus deployment that uses hashmod sharding, each shard would only send metadata from the actively scraped targets.

While this works reasonably well for Prometheus-specific architectures and is easy for the sender, it does not achieve the goal of making metadata more generally useful or easy to consume by receivers. Some examples of issues that exist are:

- Metadata does not persist after a Prometheus server restart
- Metadata is recorded per “metric family” and cannot be different for each series
- Metadata cannot be different for the same “metric family” in different targets and is being overwritten after the first target it’s seen at
- Metadata cannot track changes over time; we can only see the most recent value and only for series that are actively being scraped

- Metadata cannot be easily used by downstream systems (eg. for stream processing or aggregations) as it's not readily available at all times
- Metadata is closely coupled with the concept of Targets cannot be used by projects that bypass that concept (eg. Grafana Agent integrations)
- Current implementation of metadata in remote write requires stateful receivers (for the interval duration).
- Metadata does not persist in TSDB Block (not a goal for this proposal).

To start working around these issues, we first added Metadata as a record type in the WAL and then allowed per-series metadata to be appended to the WAL from the scrape loop where a new metadata record is being created whenever a) we see a new series, or b) metadata for an existing series changes. The most recent state of the metadata for each series is also restored on startup when the WAL is replayed.
So the final piece of the puzzle is to allow Metadata from the WAL to be sent over the remote write protocol.

This proposal outlines a new approach to propagating Metadata by adding a new field to the Remote Write protobuf definition and making metadata a first-class citizen along labels, samples, exemplars and native histograms.

The goal in mind is that users can have up-to-date metadata that is always available whenever they're interacting with a timeseries. Björn Rabenstein had expressed some similar ideas in the past in a discussion of the original work by Rob Skillington.

Proposal - Top-level metadata field

The first part of the proposal is that we add a new Protobuf field for per-series metadata that is sent as an optional field in the TimeSeries struct.

Since each series has a unique metadata definition at a certain point in time, the field is not repeated. The non-repeated field means that we're still able to express changing metadata over time, but only over separate remote_write requests. In case we get rapidly changing metadata over the course of a single remote_write request encoding, the value here will be the one last one seen when the sample is queued for (last one wins).

An implementation is already in a Draft PR in case you want to see this in code.
https://github.com/prometheus/prometheus/pull/11640/files

```
None
message Metadata {
  enum MetricType {
```

```
    UNKNOWN        = 0;
    COUNTER        = 1;
    GAUGE          = 2;
    HISTOGRAM      = 3;
    GAUGEHISTOGRAM = 4;
    SUMMARY        = 5;
    INFO           = 6;
    STATESET       = 7;
  }
  MetricType type = 1;
  string help = 2;
  string unit = 3;
}

// TimeSeries represents samples and labels for a single time
series.
message TimeSeries {
  repeated Label labels         = 1 [(gogoproto.nullable) =
false];
  repeated Sample samples       = 2 [(gogoproto.nullable) =
false];
  repeated Exemplar exemplars   = 3 [(gogoproto.nullable) =
false];
  repeated Histogram histograms = 4 [(gogoproto.nullable) =
false];
  Metadata metadata             = 5 [(gogoproto.nullable) =
false];
}
```

While that current design *did* consider about adding Metadata in-line with the TimeSeries back in Feb 2020, it opted against it citing the high overlap and deduplication of metadata that would be sent on a regular interval increasing the byte size of requests, and the possibility of slowing down ingesters and affecting shard calculations and increasing the memory footprint.

The counter-argument is that while Metadata is indeed going to increase the size of remote write requests and resource usage, it is a) typically smaller than the labels, b) its values are bounded and c) the repeatability of it will help with both current ways of addressing this (string

interning) as well as future-looking ones (a stateful remote-write with either a caching/lookup table for deduplicating metadata or sending metadata on a specific interval).

In an older PR (Aug 2020), Rob Skillington had prototyped this approach and sent metadata every 15 seconds from a node_exporter instance which led to an ~20% increase in network traffic. Once we've set on an approach, I'd like to have a prombench run that will measure the impact on today's codebase.
Adding this new field and crystallizing it into a new protocol version (as in the next section) also presents an opportunity to remove the standalone Metadata field and current mechanism so that we address the list of issues presented above.

As a footnote, the OpenTelemetry Protocol metrics schema follows a middling approach which *does* send metadata in the same request as the data points, but those metadata are still recorded per metric family.xt

Proposal - Protocol versioning

This is not a breaking change for the remote_write protocol per se, as we're appending a new field to the TimeSeries definition and this will probably be an opt-in feature for the time being.

Although, with the recent changes in the remote_write protocol that includes exemplars, and histograms as well as the ability to send metadata separately (outside the TimeSeries struct) we have already progressed past the v1.0 specification published back in April 2023.

The proposal is that we publish a new version of the remote_write protocol (v1.1) that includes

- Adding the new metadata field in-line with TimeSeries
- Marking the standalone MetricMetadata field in the WriteRequest as deprecated, and eventually disabling it via the feature flag and removing it in a future iteration.

If the decision is to go ahead with this, we will use this proposal to explore what it means for the various server implementations, and how to negotiate the protocol with them.

Sending interval

Another question that needs to be answered is how and how often the Metadata will be propagated within remote write requests.

One distinction has to do with whether metadata will be sent along *every* remote write request, similar to labels. The alternative is to have metadata be propagated whenever a new WAL record is present (that is, for new series and series whose metadata has changed), or on a specific interval (every X seconds or Y samples).

The second distinction has to do with whether metadata are sent in-line with the rest of the data points (samples/exemplars/histograms) on requests, or if they are sent on separate requests on their own.

To summarize the available options:

- **[proposed]** Included with datapoints; always
- Included with datapoints whenever a new WAL record is appended
- Included with datapoints on a specific interval
- Separated from datapoints, whenever a new WAL record is appended
- Separated from datapoints, on a specific interval.

My thinking is that separating metadata from data points is a no-go.
It gets us a step back, and instead it invalidates most of the usefulness of the proposed protobuf field. Transfer-wise, it introduces a bunch of new requests that have to re-transmit the series' set of labels, and due to the way that remote write requests are sharded and batched, metadata is not guaranteed to be available when processing a datapoint.

As to how often to send metadata, I'd like to argue that, similar to labels, metadata is a crucial part of any datapoint being sent so it should be *always* bundled with all TimeSeries structs.
This may be the most resource-intensive of the options, but it actually guarantees that metadata is available whenever any consumer is processing a timeseries record.
Their size is bounded, and the most important concern (large HELP texts) can be softened via string interning or with other technical ways in the future (eg. introducing statefulness to the remote_write protocol).

Overhead

We ran the proposed solution on a dev environment within Grafana Labs to gauge its impact on a realistic workload. Our results showed that the proposed change

- Increases the outgoing bandwidth by ~20%
- Increases the WAL size by ~17% (measured at the 1-hour mark)

At the same time, when combined with Callum Styan 's work on a interning table, the same tests resulted in ~35% reduction in outgoing request sizes *with metadata* alongside samples.

Since a sudden increase of egress costs might be surprising to users, we propose to *combine* both changes to make it into Remote Write 1.1, configurable behind a feature flag.

Alternatives Considered

- Do nothing: continue using the MetadataWatcher implementation. We believe that the unlocking the potential of metadata in the general Prometheus ecosystem and making them more generally useful outweighs the complexity we're adding here.
- Rework the MetadataWatcher to be per-series. This would both be a backwards-incompatible change, and in order to solve the outlined issues, it would need a stateful protocol.