Go Data Pipeline (gdp)

Introduction

This document describes a little demonstration package that has a small golang tool that streams protobufs to Kafka/Redpanda and then into Clickhouse.

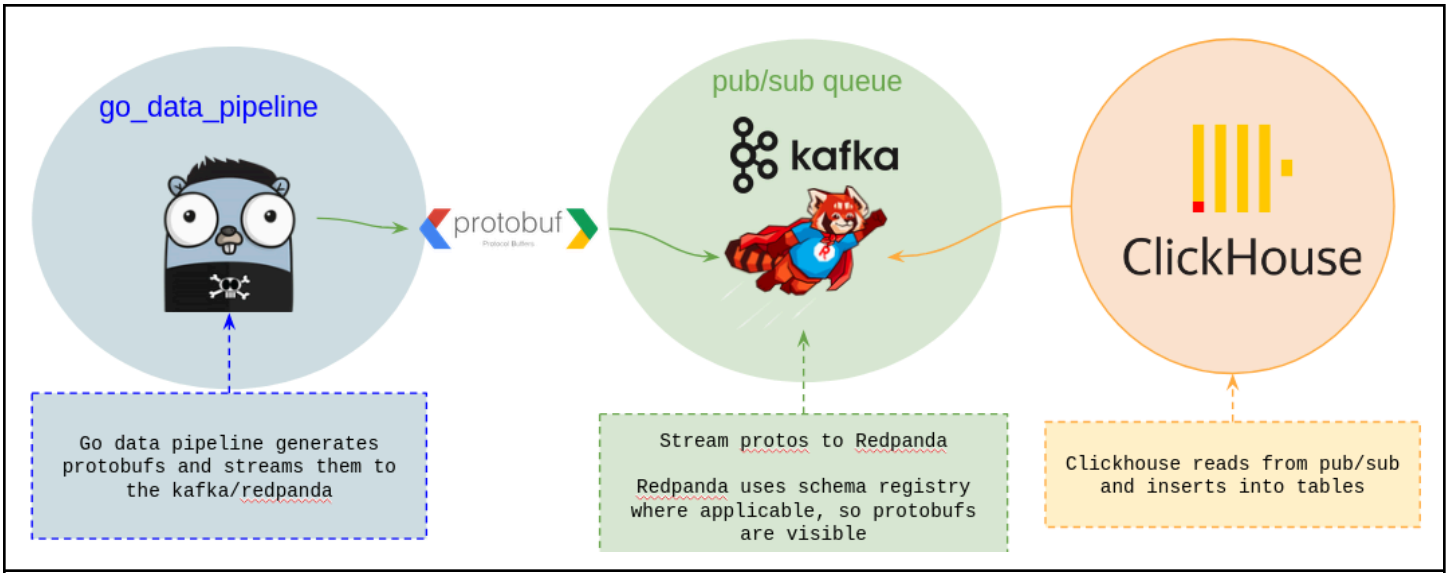
The repo's name is “gdp” , the acronym for Go Data Pipeline.
https://github.com/randomizedcoder/gdp

This is for educational purposes and is definitely not something you want to deploy into a production environment.

This repo is crazy

DO NOT USE IN PRODUCTION!!

This project mostly started out of frustration that I could not find an open source example, so I hope others find this useful. Additionally, hopefully this will help people understand the various protobuf data types supported by Clickhouse.



Go data pipeline overview

Table of Contents

Motivation

At multiple companies, we've spent a reasonable amount of time getting data pipelines working nicely. In these cases, it took time to understand the pipelines and for the engineers to understand how all the steps work. Hopefully this project will help others to understand the various trade-offs and to see the implementation details.

I'm hoping this repo will serve as a reasonable example of how to do things, so it will make implementation easier for others. This is my small part in giving back to the open source community. I'm optimistic this will help both Redpanda and Clickhouse, which are both great open source companies.

For example, a repo like this would have really increased our velocity, and I hope it speeds up yours.

Quick Start

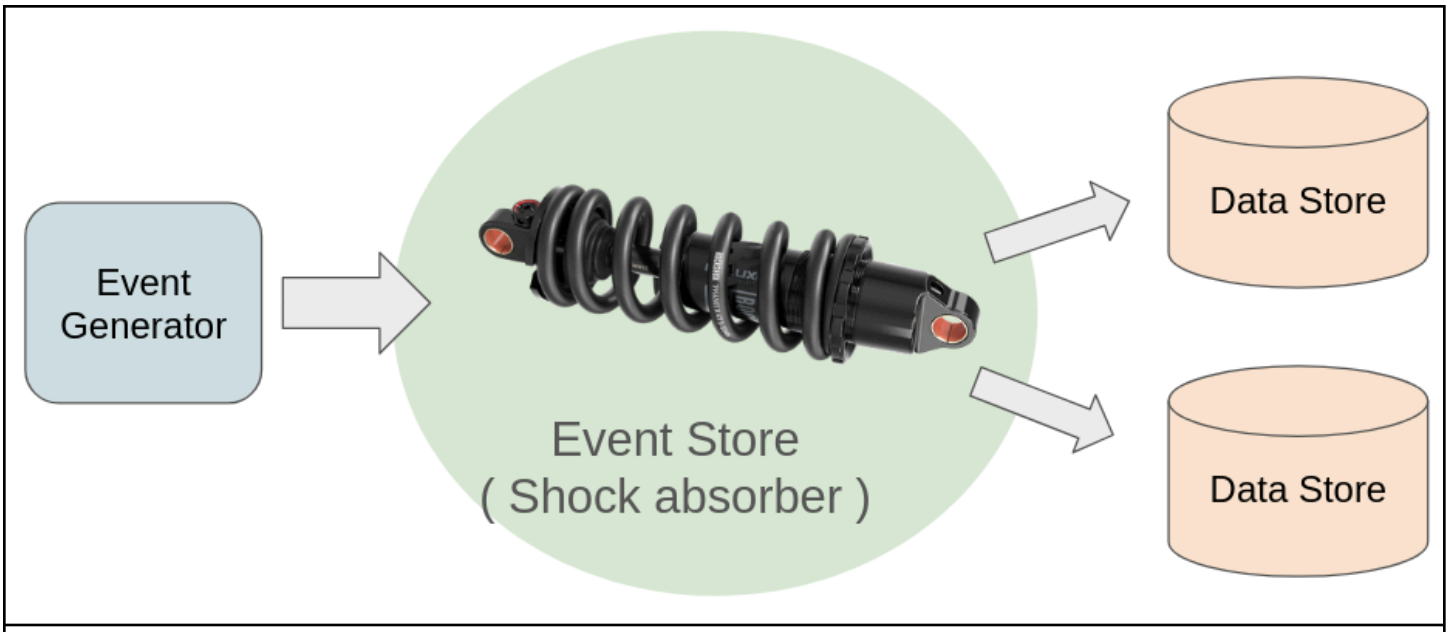
I’m hoping the repo is reasonably usable quickly

https://github.com/randomizedcoder/gdp/blob/main/README.md

Design Principles

Reliable data pipelines have three (x3) main components:

- **Event generator(s)**
    - Events are generated by some kind of state change, and/or there is polling. In this project, we're just doing simple polling.
- **Distributed temporary event store**
    - The distributed event store adds reliability and acts a little bit like a shock absorber.
        - If something goes wrong with the data store, messages will remain queued up in the event store
        - If the message rate spikes briefly, messages will also queue up in the event store
    - The event store allows for multiple consumers, which can be at different positions reading, and essentially means data can be replicated to multiple data stores.
- **Data store(s)**
    - The data stores allow for large scale online data analytic processing.
    - The data is usually for monitoring and billing.



Reliable data pipelines major components

Go Data Pipeline

Robust and efficient data pipelines are extremely important for monitoring and operation visibility. There are many implementations with various trade offs.

This repo demonstrates an example of a data pipeline, based on the following:

- **Golang**
  - Golang is a relatively popular and efficient programming language
    - https://go.dev/
  - While golang has a garbage collector (GC) it is still pretty efficient. In performance sensitive scenarios, like we have for a data pipeline where we know we will have many objects with short lifetimes, pressure on the allocator and GC can be reduced by using memory "pool" techniques (https://pkg.go.dev/sync#Pool).
    - This repo does a little bit of this, searching for ".Get(" and ".Put(". See also:
      - https://github.com/randomizedcoder/gdp/blob/main/pkg/gdp/init_sync_pools.go
- **Franz-go Kafka Client**
  - The golang tool uses the Franz-go library which is a native go implementation of a kafka client. (This is in distinct contrast to other kafka libraries that use CGO, which is of course much lower performance.)
    - https://github.com/twmb/franz-go
    - https://pkg.go.dev/github.com/twmb/franz-go/pkg/kgo
- **Protobufs (Protocol Buffers)**
  - Protobufs may not be the most efficient serialization method, but they are reasonably efficient, and very widely deployed. Protobufs are dramatically more efficient than JSON, both from a data size perspective, but more importantly from a parsing perspective. See also the section "Protobuf efficiency".
    - https://protobuf.dev/
  - In this repo we demonstrate various different protobuf encapsulation methods, and discuss the pros/cons of these. Later in this document there is more detail describing the encapsulation methods.
  - Hopefully having an open source example of the various encapsulation methods will help others understand the trade-offs
- **Buf Build**
  - This repo uses Buf to compile the protobufs. This is highly recommended compared to trying to get protoc to work.
    - https://buf.build/
  - The buf configuration builds a lot more than is required, so it builds openAPI and python, for example.
  - Please note there are also examples of the pretty nice validation that buf can do. The validation is highly recommended. We use the validation in production to help ensure all our GRPC calls are valid.
    - The validation actually uses the latest version of the buf validators which are very cool and to my knowledge not possible with other validators. They let you define validators *between* different attributes of the same protobuf. (Nice work buf team!)
      - E.g. poll_frequency > poll_timeout
      - https://github.com/randomizedcoder/gdp/blob/main/proto/gdp_config/v1/gdp_config.proto#L59
- **Kafka/Redpanda**
  - Event streaming needs to be reliable, so events are streamed to a distributed event store for reliability.
    - Reliability means that the data sink can be stopped for upgrades/maintenance, and the events will queue up in the event streaming platform, until the data sink is operational again.
    - Multiple consumers can also be set up, so that data can be stored in multiple locations for more reliability. E.g. Consume data across multiple data centers.

    - Kafka is a very popular distributed event store and stream-processing platform
        - https://en.wikipedia.org/wiki/Apache_Kafka
    - Redpanda is a Kafka compatible event store, which is a lot more efficient
        - https://www.redpanda.com/
        - https://docs.redpanda.com/current/get-started/intro-to-events/
    - Redpanda has high performance because it uses SeaStar under the hood
        - https://seastar.io/
    - Events streams we can't use include:
        - NATS does not support Jetstream, so without a reliable event bus, this is a non-starter.
            - Tracking issue: https://github.com/ClickHouse/ClickHouse/issues/39459
            - Bug bounty: https://github.com/timeplus-io/proton/issues/535#issuecomment-1918211105
            - Even a closed MR: https://github.com/timeplus-io/proton/pull/859/files
            - NATS also doesn't support schema registry
        - Buf
            - Buf has an events store capability, and it looks pretty impressive, but there's no Clickhouse integration.
- **Schema Registry**
    - Schema registries are designed to allow large organizations manage the life cycle of their schemas, which isn't really relevant to this demo repo
    - Schema registry is also a convenience feature of event streaming platforms to allow operational visibility of the data flowing through the system. E.g. Allow operators to see what's going on
        - Please note that there are other capabilities like KSQL that are possible, but this is not demonstrated at this time.
- **Online analytical processing (OLAP) / Clickhouse**
    - Clickhouse is a very high performance columnar data store
        - https://en.wikipedia.org/wiki/ClickHouse
    - Clickhouse is mostly so fast because it uses Log Structured Merge Tree
        - https://en.wikipedia.org/wiki/Log-structured_merge-tree
        - https://youtu.be/b6SI8VbcT4w
    - This repo demonstrates the different protobuf encapsulations supported, including demonstrating the ProtobufList type, which allows batching. Clickhouse perfs batching.
        - See also:
        - https://clickhouse.com/blog/clickhouse-input-format-matchup-which-is-fastest-most-efficient
        - https://www.mux.com/blog/latency-and-throughput-tradeoffs-of-clickhouse-kafka-table-engine
        - https://altinity.com/blog/kafka-engine-the-story-continues
    - The Clickhouse kafka library is this one:
        - https://github.com/confluentinc/librdkafka

Objective - Primary

The primary objective of this repository is to demonstrate how to put together a data pipeline using the tools described, and to show what does and doesn't work, and then to explain why.

Objective - Which data to stream?

To demonstrate the data pipeline requires some data to stream. At a loss for something simple to stream, the idea I came up with was to perform an experiment of streaming Prometheus metrics to Clickhouse.

The reasons streaming Prometheus metrics to Clickhouse is interesting is that Clickhouse has the attractive features:

- **Aggregations**
  - Clickhouse has powerful aggregation capabilities, which they call "materialized views". Essentially, this means you can do rolls ups over time
    - https://clickhouse.com/docs/materialized-views
- **TTL control**
  - Clickhouse allows for granular TTL control per table, so for example, we can keep 24 hours of per 1 minute aggregations, and also keep three (x3) months of hourly aggregations
- **Horizontal scaling**
  - Prometheus is great, but as it scales it gets tricky. In a past life at Edgecast, we chose Cortex (https://cortexmetrics.io/) to provide horizontal scaling.
  - Clickhouse has great horizontal scaling capabilities, including being able to configure how many replicas are created, which allows the Clickhouse query load to scale. E.g. If the query performance isn't enough, you can add more replicas and Clickhouse will shard the queries across more nodes, increasing the performance.

Therefore, I'm using this little demonstration project as an excuse to also try out using Clickhouse for Prometheus metrics. Depending on what we learn doing this, it's something we might consider for longer term retention of Prometheus metrics. I'm pretty optimistic this is going to be awesome.

> There are others in the industry are already doing some of this via OpenTelemetry:
> - https://opentelemetry.io/docs/collector/
> - https://github.com/open-telemetry/opentelemetry-collector-contrib/blob/main/exporter/clickhouseexporter/README.md
>
> This demonstration is *not* designed to compete with this, I'm just using the prometheus metrics as an experiment and to demo the data pipeline.

Objective - Prometheus Counters

Within most of the go code I write, I've settled on a partner of using Prometheus counters with three (x3) labels:

1. **Function**
   a. Function name
2. **Variable**
   a. Particular variable within the function that is being counted
3. **Type**
   a. Counter or Error
   b. The nice thing about this is that you can easily find if the code is generating errors

The key point is that there are only ever x3 labels, which means it is easy to map these to Clickhouse columns.

The prometheus counter is defined like this:

```
x.pC = promauto.NewCounterVec(
  prometheus.CounterOpts{
        Subsystem: "xtcp",
        Name:  "counts",
        Help:  "xtcp counts",
  },
  []string{"function", "variable", "type"},
```

```
x.pH = promauto.NewSummaryVec(
  prometheus.SummaryOpts{
        Subsystem: "xtcp",
        Name:  "histograms",
        Help:  "xtcp histograms",
        Objectives: map[float64]float64{
        0.1:  quantileError,
        0.5:  quantileError,
        0.99: quantileError,
        },
        MaxAge: summaryVecMaxAge,
  },
  []string{"function", "variable", "type"},

// Please note that the gdp repo does NOT yet implement the histograms. Maybe I'll add this soon.
```

Then within the go code the counters can be easily sprinkled throughout the code. This means that when looking at the Prometheus metrics, it is really easy to see what's going on, and what's generating errors. I find this approach a lot easier than looking at logs.

```
startTime := time.Now()
defer func() {
  d.pH.WithLabelValues("Deserialize", "complete", "count").Observe(time.Since(startTime).Seconds())
}()
d.pC.WithLabelValues("Deserialize", "start", "count").Inc()
```

```
// Error counters
d.pC.WithLabelValues("Deserialize", "netlinkerDoneCh", "error").Inc()
```

To find any errors being generated is then as simple as doing:

```
curl -s http://my.go.application:8888/metrics | grep error
```

```
Finding error counters
```

Within the gdp code, an example of using both the counters and the histograms is shown here. These counters are tracking the number of messages sent per topic, the number of bytes sent per topic. The histograms are primarily for recording timing, so for example, how long it took to send the message to Kafka. These are obviously all very useful metrics to understand when trying to reason about the behavior of the system.

```
g.pH.WithLabelValues("destKafka", mc.Topic, "count").Observe(dur.Seconds())
g.pC.WithLabelValues("destKafka", mc.Topic, "count").Inc()
g.pC.WithLabelValues("destKafka", mc.Topic, "n").Add(float64(n))
```

https://github.com/randomizedcoder/gdp/blob/main/pkg/gdp/destinations.go#L95C1-L97C68

Here is an example of the counts from the gdp after it has been running overnight. We can see there are counters for each type of Protobuf that's been sent to kafka by the function "destKafka", for example. We can also see that there are no counters with the label "error", so nothing has gone wrong (yay).

```
[das@hp1:~/gdp]$ curl -s http://localhost:8888/metrics 2>&1 | grep -v '#' | grep counts
gdp_counts{function="Poller",type="count",variable="pollingLoops"} 24234
gdp_counts{function="Poller",type="count",variable="ticker"} 24233
gdp_counts{function="Run",type="counter",variable="start"} 1
gdp_counts{function="destKafka",type="count",variable="Protobuf"} 872357
gdp_counts{function="destKafka",type="count",variable="ProtobufKafka"} 872357
gdp_counts{function="destKafka",type="count",variable="ProtobufKafkaProtodelim"} 872357
gdp_counts{function="destKafka",type="count",variable="ProtobufList"} 24233
gdp_counts{function="destKafka",type="count",variable="ProtobufListKafka"} 24233
gdp_counts{function="destKafka",type="count",variable="ProtobufListKafkaProtodelim"} 24233
gdp_counts{function="destKafka",type="count",variable="ProtobufListProtodelim"} 24233
gdp_counts{function="destKafka",type="count",variable="ProtobufProtodelim"} 872357
gdp_counts{function="destKafka",type="count",variable="ProtobufSingle"} 872357
gdp_counts{function="destKafka",type="count",variable="ProtobufSingleKafka"} 872357
gdp_counts{function="destKafka",type="count",variable="ProtobufSingleKafkaProtodelim"} 872357
gdp_counts{function="destKafka",type="count",variable="ProtobufSingleProtodelim"} 872357
gdp_counts{function="destKafka",type="count",variable="start"} 7.075788e+06
gdp_counts{function="destKafka",type="n",variable="Protobuf"} 6.9509151e+07
gdp_counts{function="destKafka",type="n",variable="ProtobufKafka"} 7.4743293e+07
gdp_counts{function="destKafka",type="n",variable="ProtobufKafkaProtodelim"} 7.561565e+07
gdp_counts{function="destKafka",type="n",variable="ProtobufList"} 7.1253865e+07
gdp_counts{function="destKafka",type="n",variable="ProtobufListKafka"} 7.1399263e+07
gdp_counts{function="destKafka",type="n",variable="ProtobufListKafkaProtodelim"} 7.1447729e+07
gdp_counts{function="destKafka",type="n",variable="ProtobufListProtodelim"} 7.1302331e+07
gdp_counts{function="destKafka",type="n",variable="ProtobufProtodelim"} 7.0381508e+07
gdp_counts{function="destKafka",type="n",variable="ProtobufSingle"} 6.9509151e+07
gdp_counts{function="destKafka",type="n",variable="ProtobufSingleKafka"} 7.4743293e+07
gdp_counts{function="destKafka",type="n",variable="ProtobufSingleKafkaProtodelim"} 7.561565e+07
gdp_counts{function="destKafka",type="n",variable="ProtobufSingleProtodelim"} 7.0381508e+07
gdp_counts{function="fetchPrometheusMetrics",type="count",variable="bodyBytes"} 4.77237935e+08
gdp_counts{function="fetchPrometheusMetrics",type="count",variable="firstShot"} 24233
gdp_counts{function="fetchPrometheusMetrics",type="count",variable="start"} 24233
gdp_counts{function="fetchPrometheusMetrics",type="count",variable="success"} 24233
gdp_counts{function="filterCounts",type="count",variable="counts"} 872357
gdp_counts{function="filterCounts",type="count",variable="filtered"} 5.016149e+06
gdp_counts{function="performPoll",type="count",variable="start"} 24233
gdp_counts{function="sendEnvelopeWithMarshalConfig",type="count",variable="start"} 290796
```

```
[das@hp1:~/gdp]$ curl -s http://localhost:8888/metrics 2>&1 | grep -v '#' | grep counts | grep error

(yay no errors!)
```

Please note that gdp does not yet handle the histograms.

Objective - Prometheus Protobuf

Mapping the prometheus counters to a protobuf is pretty straight forward.

Features worth noting:

- **Time**
  - Clickhouse only supports 64bit time columns, so we're going to use 64bits
    - https://clickhouse.com/docs/sql-reference/data-types/datetime64
  - This resolution is surplus to requirements, given that we intend to sample the metrics in the order of seconds
  - Linux time is higher resolution, but we don't ended this:
    - https://github.com/torvalds/linux/blob/master/include/linux/time64.h#L13
- **Annotations**
  - I've included several string fields to allow details about the metrics source to be annotated (PoP, label, tag).
- **Poll and record counters**

- To allow verification of integrity of the data pipeline, but including a poll_counter and record_counter, it will be easy to detect if the data pipeline is missing samples.

Please note that a key feature of the prometheus collection philosophy is to always collect the counter and NOT a delta.

This way, if the sampling frequency changes, and/or samples are lost, the counter remains accurate.

E.g. Assuming the metric being sampled is for billing bytes, for example, and if you are sampling every minute, and there's an issue in the data pipeline, it's less of a concern if data is dropped for a few samples, because you will still be able to bill for all the bytes, even if the resolution is reduced.

The simplest form of the proto is this:

```
syntax = "proto3";

package prometheus.v1;

option go_package = "./pkg/prometheus";

message PromRecordCounter {
  double timestamp_ns = 10;
  string hostname = 20;
  string pop = 30;
  string label = 50;
  string tag = 60;
  uint64 poll_counter = 70;
  uint64 record_counter = 80;
  string function = 90; // []string{"function", "variable", "type"},
  string variable = 100;
  string type = 110;
  double value = 120;
}
```

prometheus.proto

Clickhouse has a much higher performance if inserts are performed in batches, so I've also defined a Clickhouse protobufList version (https://clickhouse.com/docs/interfaces/formats/ProtobufList). Please note that Clickhouse needs the row defined within the "Envelope", which is apparently a limitation of the Clickhouse protobuf parsing/compiling capabilities.

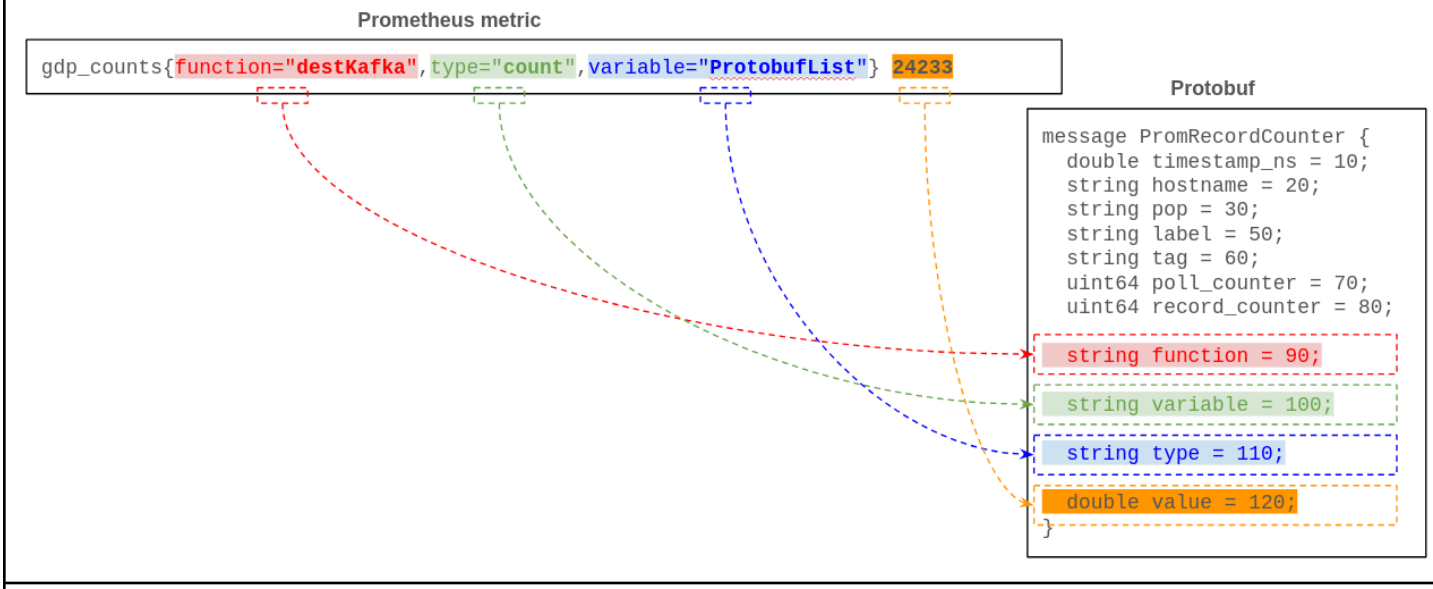
```
syntax = "proto3";

package prometheus.v1;

option go_package = "./pkg/prometheus";

message Envelope {
  message PromRecordCounter {
        double timestamp_ns = 10;
        string hostname = 20;
        string pop = 30;
        string label = 50;
        string tag = 60;
        uint64 poll_counter = 70;
        uint64 record_counter = 80;
        string function = 90; // []string{"function", "variable", "type"},
        string variable = 100;
        string type = 110;
        double value = 120;
  }
  repeated .prometheus.v1.Envelope.PromRecordCounter rows = 1;
}

message PromRecordCounter {
```

```
  double timestamp_ns = 10;
  string hostname = 20;
  string pop = 30;
  string label = 50;
  string tag = 60;
  uint64 poll_counter = 70;
  uint64 record_counter = 80;
  string function = 90; // []string{"function", "variable", "type"},
  string variable = 100;
  string type = 110;
  double value = 120;
}
```

Prometheus_protolist.proto. Highlighted in blue is the "inner" message, which needs to be defined here, or Clcikhouse doesn't recognise the row correctly.

Objective - Prometheus Protobuf - Diagram

Visually we can see that the prometheus metrics map directly into the protobuf as shown here:



Prometheus to protobuf mapping

Objective - Clickhouse Table

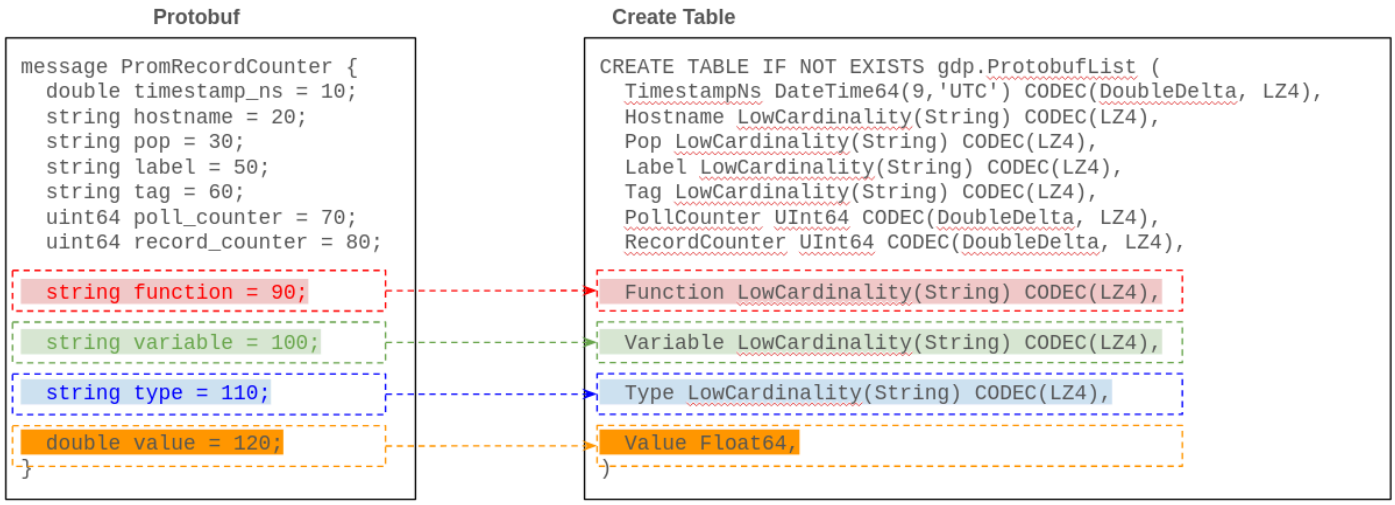
The protobuf can then be mapped to columns in a clickhouse table.

This is an example of the table creation SQL.

```
DROP TABLE IF EXISTS gdp.ProtobufList;

CREATE TABLE IF NOT EXISTS gdp.ProtobufList (
  Timestamp_Ns DateTime64(9,'UTC') CODEC(DoubleDelta, LZ4),
  Hostname LowCardinality(String) CODEC(LZ4),
  Pop LowCardinality(String) CODEC(LZ4),
  Label LowCardinality(String) CODEC(LZ4),
  Tag LowCardinality(String) CODEC(LZ4),
  Poll_Counter UInt64 CODEC(DoubleDelta, LZ4),
  Record_Counter UInt64 CODEC(DoubleDelta, LZ4),
  Function LowCardinality(String) CODEC(LZ4),
  Variable LowCardinality(String) CODEC(LZ4),
```

```
  Type LowCardinality(String) CODEC(LZ4),
  Value Float64,
)
ENGINE = MergeTree()
PARTITION BY toYYYYMM(TimestampNs)
ORDER BY (TimestampNs, Hostname, Pop, Label, Tag, PollCounter, RecordCounter)
TTL toDateTime(TimestampNs) + INTERVAL 14 DAY;
```

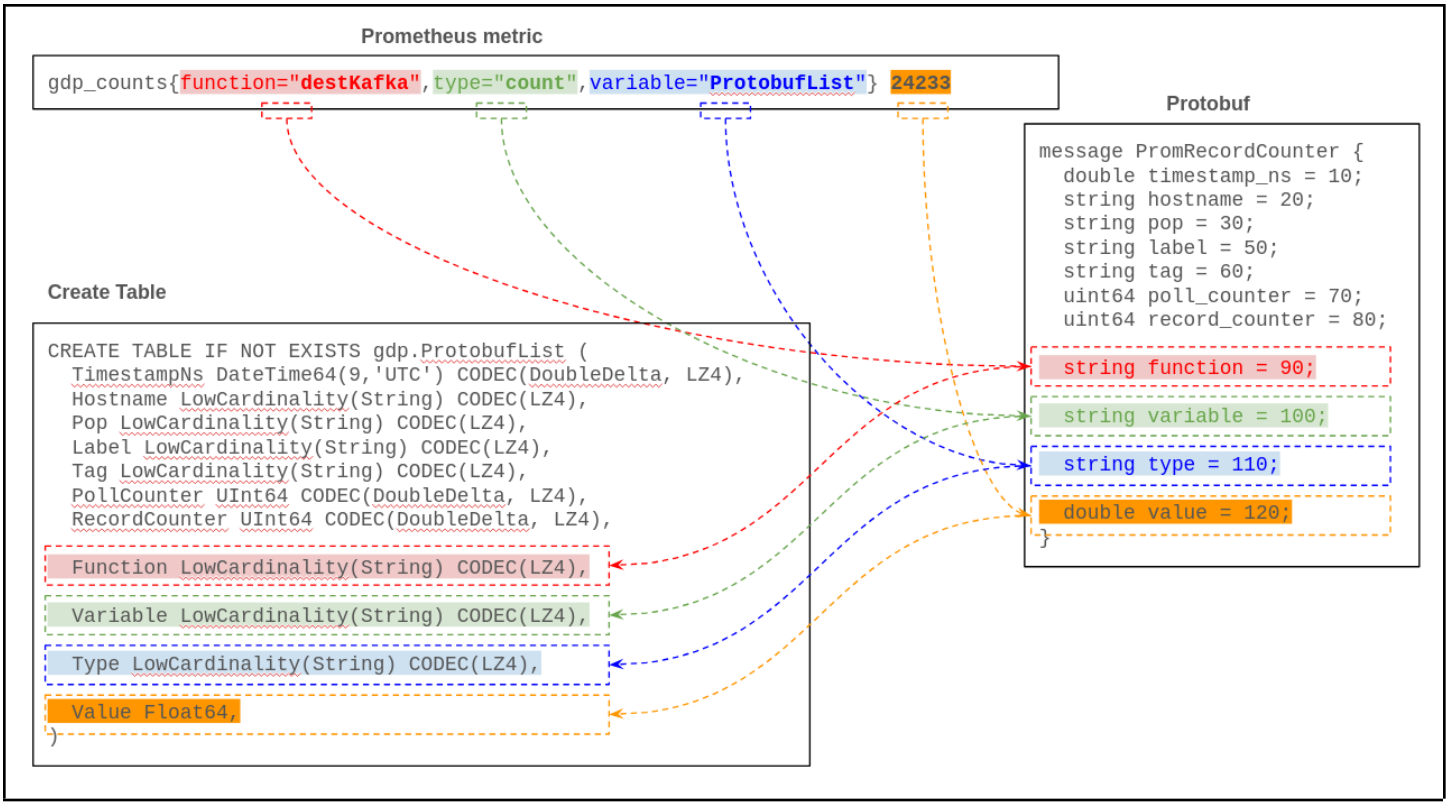
Example of the table creation SQL



Protobuf to Clickhouse table mapping

Objective - Overall metrics flow

Bit of a messy diagram, but this attempts to depict the flow of the data from the prometheus metric source, to the protobuf, and then finally into the clickhouse table.



Protobuf efficiency

To put the efficiency of protobuf versus JSON into perspective, here is a simple comparison of the binary wire encoded protobuf size versus the JSON size in bytes. Obviously, the exact sizes will vary slightly based on the data, but this should give you a rough sense of the order of magnitude we are talking about. This doesn't take into account compression, where we would expect the JSON to kafka zstd compress pretty well, but fundamentally when both Redpanda and Clickhouse are processing the data, they are handling the uncompressed data.

The table shows that the binary encoded protobuf is roughly x2 more efficient and JSON

	Wire size (bytes)	**JSON size (bytes)**	**Efficiency**
Protobuf	82	182	82/182 ~= 0.450
ProtobufList	2935	6933	2935/6933 ~= 0.423

The marshalled single entry is 82 bytes, while the JSON encapsulated version is 182 bytes in this case.

```
2025/04/02 18:06:19.402975 gdp_kafka_client.go:110: len(record.Value):82, schemaID:6, protobufSchemaIndex:
0, header:00 00 00 00 06 00
2025/04/02 18:06:19.402988 gdp_kafka_client.go:143: printPayload row JSON:
{"timestampNs":1743617179.389086,"hostname":"071850d7c92f","pollCounter":"27067","recordCounter":"33","fun
ction":"filterCounts","variable":"filtered","type":"count","value":5602787}
```

Protobuf versus protobuf as JSONS

The marshalled protobufList entry is 2935 bytes, while the JSON encapsulated version is 6933 bytes in this case.

```
2025/04/02 18:19:27.650203 gdp_kafka_client.go:110: len(record.Value):2935, schemaID:4,
protobufSchemaIndex: 0, header:00 00 00 00 04 00
2025/04/02 18:19:27.650754 gdp_kafka_client.go:160: printPayload envelope JSON:
{"rows":[{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892",
"function":"Poller", "variable":"pollingLoops", "type":"count", "value":2893},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"1",
"function":"Poller", "variable":"ticker", "type":"count", "value":2893},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"2",
"function":"Run", "variable":"start", "type":"counter", "value":1}, {"timestampNs":1743568829.3891041,
"hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"3", "function":"destKafka",
"variable":"Protobuf", "type":"count", "value":104081}, {"timestampNs":1743568829.3891041,
"hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"4", "function":"destKafka",
"variable":"ProtobufKafka", "type":"count", "value":104081}, {"timestampNs":1743568829.3891041,
"hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"5", "function":"destKafka",
"variable":"ProtobufKafkaProtodelim", "type":"count", "value":104081}, {"timestampNs":1743568829.3891041,
"hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"6", "function":"destKafka",
"variable":"ProtobufList", "type":"count", "value":2892}, {"timestampNs":1743568829.3891041,
"hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"7", "function":"destKafka",
"variable":"ProtobufListKafka", "type":"count", "value":2892}, {"timestampNs":1743568829.3891041,
"hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"8", "function":"destKafka",
"variable":"ProtobufListKafkaProtodelim", "type":"count", "value":2892},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"9",
"function":"destKafka", "variable":"ProtobufListProtodelim", "type":"count", "value":2892},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892",
"recordC^Counter":"10", "function":"destKafka", "variable":"ProtobufProtodelim", "type":"count",
"value":104081}, {"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892",
"recordCounter":"11", "function":"destKafka", "variable":"ProtobufSingle", "type":"count",
"value":104081}, {"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892",
"recordCounter":"12", "function":"destKafka", "variable":"ProtobufSingleKafka", "type":"count",
"value":104081}, {"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892",
"recordCounter":"13", "function":"destKafka", "variable":"ProtobufSingleKafkaProtodelim", "type":"count",
"value":104081}, {"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892",
"recordCounter":"14", "function":"destKafka", "variable":"ProtobufSingleProtodelim", "type":"count",
"value":104081}, {"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892",
"recordCounter":"15", "function":"destKafka", "variable":"start", "type":"count", "value":844216},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"16",
"function":"destKafka", "variable":"Protobuf", "type":"n", "value":8255350},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"17",
"function":"destKafka", "variable":"ProtobufKafka", "type":"n", "value":8879836},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"18",
"function":"destKafka", "variable":"ProtobufKafkaProtodelim", "type":"n", "value":8983917},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"19",
"function":"destKafka", "variable":"ProtobufList", "type":"n", "value":8463512},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"20",
"function":"destKafka", "variable":"ProtobufListKafka", "type":"n", "value":8480864},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"21",
"function":"destKafka", "variable":"ProtobufListKafkaProtodelim", "type":"n", "value":8486648},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"22",
"function":"destKafka", "variable":"ProtobufListProtodelim", "type":"n", "value":8469296},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"23",
"function":"destKafka", "variable":"ProtobufProtodelim", "type":"n", "value":8359431},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"24",
"function":"destKafka", "variable":"ProtobufSingle", "type":"n", "value":8255350},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"25",
"function":"destKafka", "variable":"ProtobufSingleKafka", "type":"n", "value":8879836},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"26",
"function":"destKafka", "variable":"ProtobufSingleKafkaProtodelim", "type":"n", "value":8983917},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"27",
"function":"destKafka", "variable":"ProtobufSingleProtodelim", "type":"n", "value":8359431},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"28",
"function":"fetchPrometheusMetrics", "variable":"bodyBytes", "type":"count", "value":56728017},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"29",
"function":"fetchPrometheusMetrics", "variable":"firstShot", "type":"count", "value":2892},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"30",
"function":"fetchPrometheusMetrics", "variable":"start", "type":"count", "value":2893},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"31",
"function":"fetchPrometheusMetrics", "variable":"success", "type":"count", "value":2892},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"32",
"function":"filterCounts", "variable":"counts", "type":"count", "value":104081},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"33",
"function":"filterCounts", "variable":"filtered", "type":"count", "value":598562},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"34",
```

"function":"performPoll", "variable":"start", "type":"count", "value":2893},
{"timestampNs":1743568829.3891041, "hostname":"071850d7c92f", "pollCounter":"2892", "recordCounter":"35",
"function":"sendEnvelopeWithMarshalConfig", "variable":"start", "type":"count", "value":34704}]}

Protobuf versus protobuf as JSONS

Clickhouse Protobuf Types

Clickhouse supports multiple types of protobufs

- **Protobuf**
  - https://clickhouse.com/docs/interfaces/formats/Protobuf
  - This was the first Clickhouse supported type of Protobuf. The challenging thing about this is that the protobufs need to be "length delimited", which essentially means there is varint length header before the protobuf payload. To marshal to this version of protobuf use the protodelim MarshalTo method
    - https://pkg.go.dev/google.golang.org/protobuf@v1.36.3/encoding/protodelim#MarshalTo
  - The unfortunate thing about this "Protobuf" type is that is largely **incompatible** with the reset of the entire Protobuf ecosystem
    - This is why Redpanda can't recognise these protobufs, as will be shown later in the document
  - Obviously, the reason Clickhouse implemented this form was so that when processing a stream of data encoded in this type, they know how much to read, like many network protocols with type-length-value (TLV). Ironically, when you want to use Protobufs with Kafka, Kafka is obviously message based, so there's not really a need for the length, because that's built into the Kafka protocol.
  - Potentially a better name for this type would have been "Protodelim"

Varint Length of payload	Encoded Protobuf Payload (this is the "normal" binary payload of the marshalled protobuf)

- **ProtobufSingle**
  - https://clickhouse.com/docs/interfaces/formats/ProtobufSingle
  - This protobuf type is the closest thing to protobufs used by the rest of the protobuf ecosystem. Most people marshal to this type already.
  - This protobuf type was added by Kenji.
  - The difference between ProtobufSingle and Protobuf is that ProtobufSingle does NOT have the length before the payload. It just has the payload.

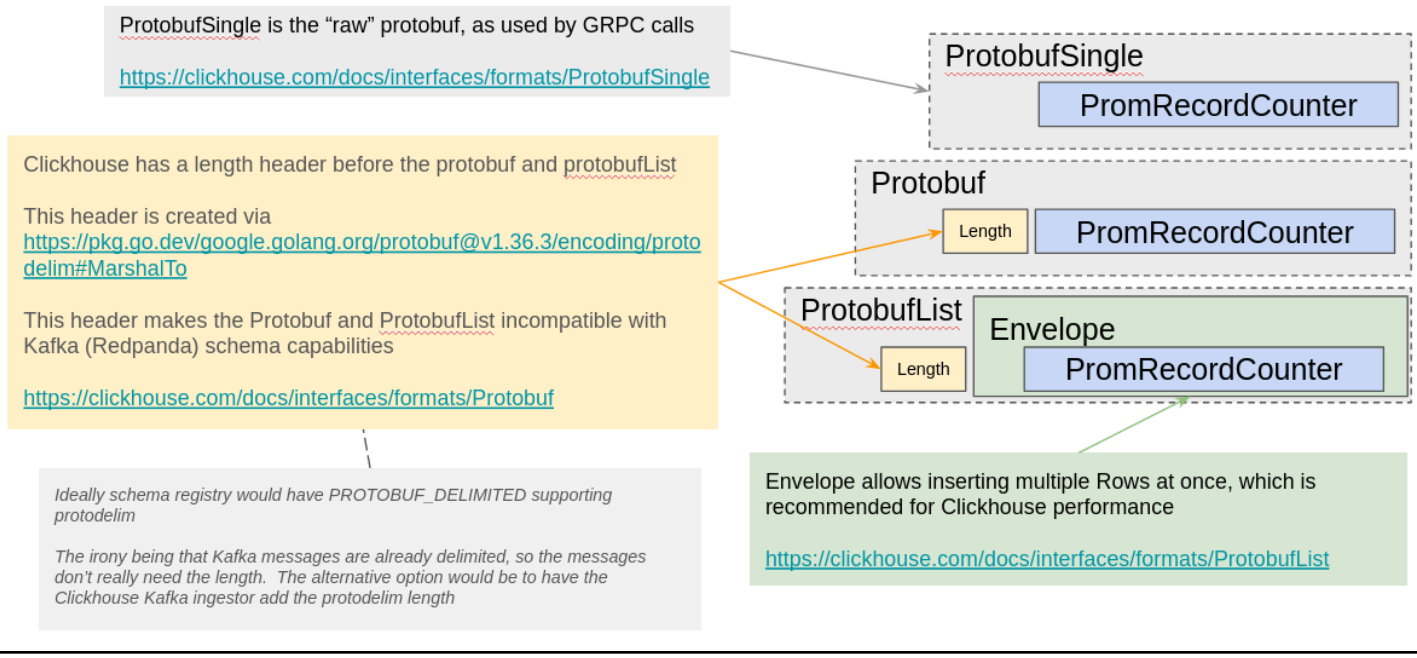
<nothing>	Encoded Protobuf Payload (this is the "normal" binary payload of the marshalled protobuf)

- **ProtobufList**
  - https://clickhouse.com/docs/interfaces/formats/ProtobufList
  - ProtobufList allows for batching of multiple "rows", so this type is a lot more efficient for inserting data into Clickhouse.
  - Unfortunately, this type is confusing to use:
    - because the Clickhouse protobuf parser is pretty limited. You apparently need to use an embedded inner message type for the row
    - And when referring to the protobuf for the Kafka table engine, you need to specify the name of the inner message type, but NOT like you would in golang. For example, the inner message in golang has the name OuterMessage_InnerMessage, but in Clickhouse you need to refer to just InnerMessage.
  - Protobuf list was added via this PR: https://github.com/ClickHouse/ClickHouse/pull/35152

Varint	Envelope	

Length of the entire Envelope		
		Encoded Protobuf Payload (this is the “normal” binary payload of the marshalled protobuf)
		Encoded Protobuf Payload (this is the “normal” binary payload of the marshalled protobuf)

The following diagram attempts to represent the different types graphically



Visual representation of the multiple Protobuf types supported by Clickhouse

What’s in a name? Protobuf name nightmares

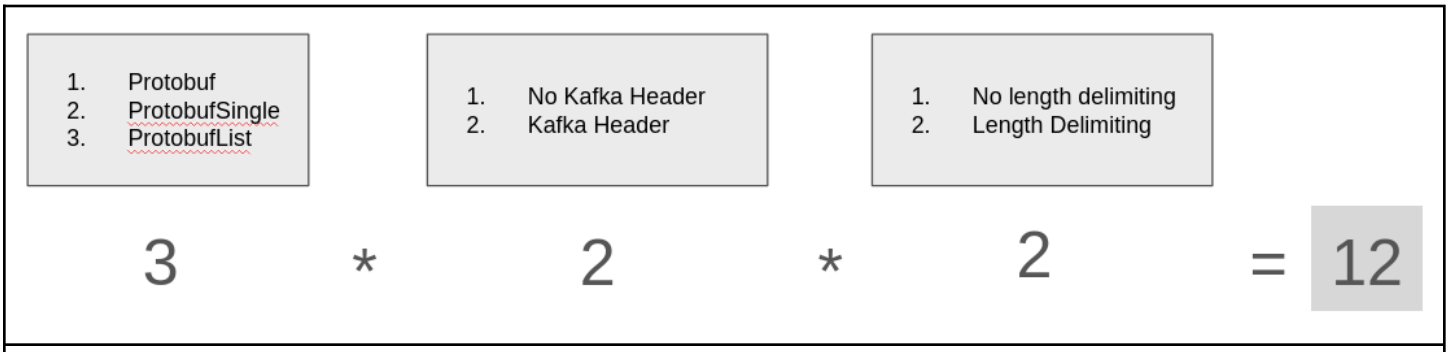
Protobufs were defined by C++ programmers, so they used c/c++ coding conventions = snake_case. Golang uses CamelCase, and Clickhouse being C++ also uses snake_case, but Clickhouse is case insensitive.

Processor	**Case styles / Naming Convention**	**Case sensitive**	**Example**
Protobuf/protoc/buf	snake_case	Yes	timestamp_ns
Golang	CamelCase	Yes	TimestampNs
Clickhouse protobuf	snake_case	No	timestamp_ns or Timestamp_Ns

Combinations and Permutations

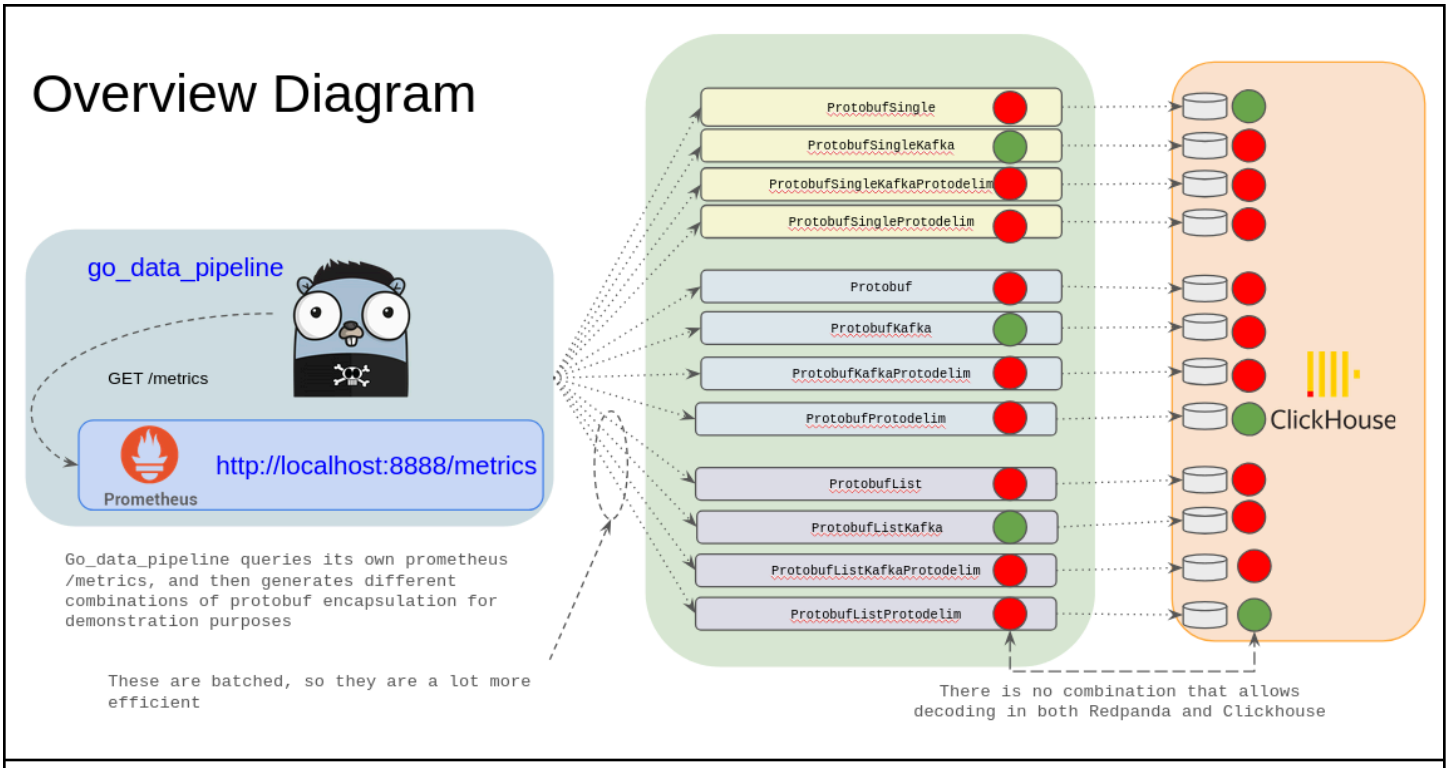
To demonstrate the possible options, this repo implements the various combinations and permutations of the:

- Clickhouse supported protobuf types
  - Protobuf
  - ProtobufSingle, and
  - ProtobufList
- Kafka header
  - No Kafka header or,
  - Kafka header
- Protobuf Length delimited
  - No length limiting
  - VarInt length before the actually binary protobuf ot not



Combinations and permutations possible with this repo

Diagrammatically the combinations and permutations looks like this:



Combinations and permutations

Ideal Combinations

The ideal combinations to have the "best" solution would be:

1. Event generators could use:
    a. Standard protobuf Marshalling, with no length header
    b. Kafka header, so that Redpanda can see the protobufs via the schema registry
    c. Envelopes, so that multiple rows could be batched. This is better for overall performance, and a lot better for Clickhouse.
    d. This would be pretty standard usage for any event generator.
2. Event Store
    a. Redpanda would be able to decode the protobufs, so that would be great
    b. No change required here.
3. Clickhouse
    a. Clickhouse could then read the Kafka messages, with the Kafka header, ***and* without** the length delimiting header

While Clickhouse requires the length delimiting the best we can hope for is that ProtobufList works, such that batch inserts work well.

Expectations - What do we expect to work today?

Today, there are actually very few combinations that will end up with data in Clickhouse.

Clickhouse will only work when:

- No kafka header
- Length delimited for Protobuf and ProtobufList
- Not length delimited for ProtobufSingle
- ProtobufList is the only option that does batch inserts

#	Marshal type	Kafka Header	Proto delimited length header	Kafka Schema Works?*	Clickhouse Insert Works?	Clickhouse batch insert (Envelope)?	Comment
0	**ProtobufSingle**	No	No	Yes	Yes	No	
1		Yes	No	Yes	No	No	
2	**Protobuf**	No	Yes	No	Yes	No	
3		Yes	Yes	No	No	No	
4	**ProtobufList**	No	No	No	No	No	
5		No	Yes	No	Yes	Yes	
6		Yes	Yes	No	No	Yes	

GDP - Building the combinations

Within the gdp code the combinations are built up within the InitMarshalConfigs function within the marshal_configs.go file
https://github.com/randomizedcoder/gdp/blob/main/pkg/gdp/marshal_configs.go#L20

If you want to adjust the combinations, do that, and then do "make restart_gdp". This will destroy the Redpanda storage, rebuild gdp, and start them back up again. This will clear all the messages from the topics, and clear the schema registry, so that gdp can re-register proto schema for the first time.

For example, you can adjust which of the protobuf types are used here:

```
	marshalTypes := []string{
	"ProtobufSingle",
	//"Protobuf",
	"ProtobufList",
	}
```

https://github.com/randomizedcoder/gdp/blob/main/pkg/gdp/marshal_configs.go#L20C1-L24C3

The optionally filter the kafka or delimiting here:

```
		//if kafka && !delim {
		//if kafka || delim {
```

https://github.com/randomizedcoder/gdp/blob/main/pkg/gdp/marshal_configs.go#L39C1-L40C26

GDP - Marshalling the protobuf combinations

Within gdp, the function to marshal a protobuf takes an additional argument which is the details of which headers and delimiting to include. The function marshal within the marshallers.go is where this happens. Essentially, the marshal function can selectively apply the Kafka header, and switch between standard protobuf marshalling, or to use the length delimiting.
https://github.com/randomizedcoder/gdp/blob/main/pkg/gdp/marshallers.go#L40

There are a couple of tests using data exported from clickhouse, which allowed me to verify that the marshal code generate the same binary as Clickhouse.
https://github.com/randomizedcoder/gdp/blob/main/pkg/gdp/marshallers_test.go
With the test data files in this folder:
https://github.com/randomizedcoder/gdp/tree/main/pkg/gdp/test_data

To run the tests do the following:

```
[das@t:~/Downloads/gdp]$ cd pkg/gdp

[das@t:~/Downloads/gdp/pkg/gdp]$ go test -v -run TestMarshal
=== RUN   TestMarshal
=== RUN   TestMarshal/ProtobufListProtodelim
=== RUN   TestMarshal/ProtobufListProtodelim_2
--- PASS: TestMarshal (0.00s)
        --- PASS: TestMarshal/ProtobufListProtodelim (0.00s)
        --- PASS: TestMarshal/ProtobufListProtodelim_2 (0.00s)
PASS
ok      github.com/randomizedcoder/gdp/pkg/gdp  0.003s
```

Running the marshalling tests

Additionally, there's a small tool used to generate the binary protobufs, that I used this to compare what comes out of Clickhouse, to what I'm generating, and to make sure I can decode the binary messages correctly.
https://github.com/randomizedcoder/gdp/blob/main/cmd/build_binary_payloads/build_binary_payloads.go

The tool is very simple, it can either read or write a protobuf, and you can tell it the type. The tool assume the protobuf is length delimited. (To be honest, I wrote this little tool, and then turn this into the tests above.)

```
	of := flag.String("of", outFolderCst, "output folder")
	f := flag.String("f", fCst, "file name")
	w := flag.Bool("w", false, "write")
	r := flag.Bool("r", false, "read rows")
	d := flag.Uint("d", debugLevelCst, "debug level")
```

Cli flags for build_binary_payloads
https://github.com/randomizedcoder/gdp/blob/main/cmd/build_binary_payloads/build_binary_payloads.go#L41C1-L45C51

GDP - Schema Registration

The gdp code to register the protobufs happens here:
https://github.com/randomizedcoder/gdp/blob/main/pkg/gdp/schema_registry.go#L16

Note that gdp will only try to register the combinations that have "kafka", because this option means the protobuf will be encoded with the kafka header, and so is the only option that will mean Redpanda will attempt to decode the protobuf.

GDP - Sending all the different combinations

The craziest part of GDP is that it will send the same data to *all* the different combinations and permutations. This happens from with the performPoll function within the poller.go. (Maybe if you wanted to use something like gdp in a production scenario, you would change this section of the poller, to just do the combination you wanted, and performance may not be too bad.)

https://github.com/randomizedcoder/gdp/blob/main/pkg/gdp/poller.go#L164

GDP Kafka client

There's a small bit of code that can read from a Kafka topic and print out the protobuf. This tool is useful verify that the protobufs written to the topics with the various configurations. The tool has cli flags to configure the various combinations and permutations as required to read the messages from Kafka.

```
	topic := flag.String("topic", topicCst, "topic")
	group := flag.String("group", groupIDCst, "consumer group")
	pb := flag.String("pb", pbCst, "protobuf type, row/envelope")
	k := flag.Bool("k", kafkaHeaderCst, "kafka header")
	delim := flag.Bool("delim", delimCst, "protobuf delim")
```

Gdp_kafka_client cli args
https://github.com/randomizedcoder/gdp/blob/main/cmd/gdp_kafka_client/gdp_kafka_client.go#L42

https://github.com/randomizedcoder/gdp/blob/main/cmd/gdp_kafka_client/gdp_kafka_client.go

The Makefile for gdp_kafka_client has examples of instantiating the program with different arguments.

- "make envelope" or "make row" to read
- "Envelope_delim" to read the length delimited envelopes

```
envelope:
	./gdp_kafka_client -topic ProtobufListKafka -pb envelope

row:
	./gdp_kafka_client -topic ProtobufKafka -pb row

envelope_delim:
	./gdp_kafka_client -topic ProtobufListProtodelim -pb envelope -k=false -delim=true
```

Gdp_kafka_client example instantiation
https://github.com/randomizedcoder/gdp/blob/main/cmd/gdp_kafka_client/Makefile#L29

Clickhouse Database Schemas

The clickhouse database schemas are all generated by a little bit of code here:
https://github.com/randomizedcoder/gdp/blob/main/cmd/create_database_schema/create_database_schema.go

This creates x4 SQL files per table type:

- .sql creates the target table
- _kafka.sql has the command to create the Kafka engine table
- _mv.sql creates the materialized view to copy the rows from kafka to the target table
- _insert.sql is an example of inserting records into the table directly within clickhouse. This is helpful if you want to then export the records from clickhouse to verify the encoding.

The comments generated at the end of each .sql file are there to help you copy and paste useful commands to verify creation works as expected.

When within the /gdp/cmd/create_database_schema/ directory, run "make rerun" to regenerate the SQL, if you've made a change.
https://github.com/randomizedcoder/gdp/blob/main/cmd/create_database_schema/Makefile#L31

The .sql files all end up in this folder
https://github.com/randomizedcoder/gdp/tree/main/build/containers/clickhouse/initdb.d/sql/gdp

To try to create all the tables there's a script 045_recreate_gdp_tables.sh. Please note this script has some filtering, which allows you to adjust which of the combinations to try to create.
https://github.com/randomizedcoder/gdp/blob/main/build/containers/clickhouse/initdb.d/disabled/045_recreate_gdp_tables.sh

To retest the scripts, I normally do "make build_clickhouse_and_deploy", which gets to back to a blank Clickhouse, and then you can try to execute some or all of the table creation scripts.

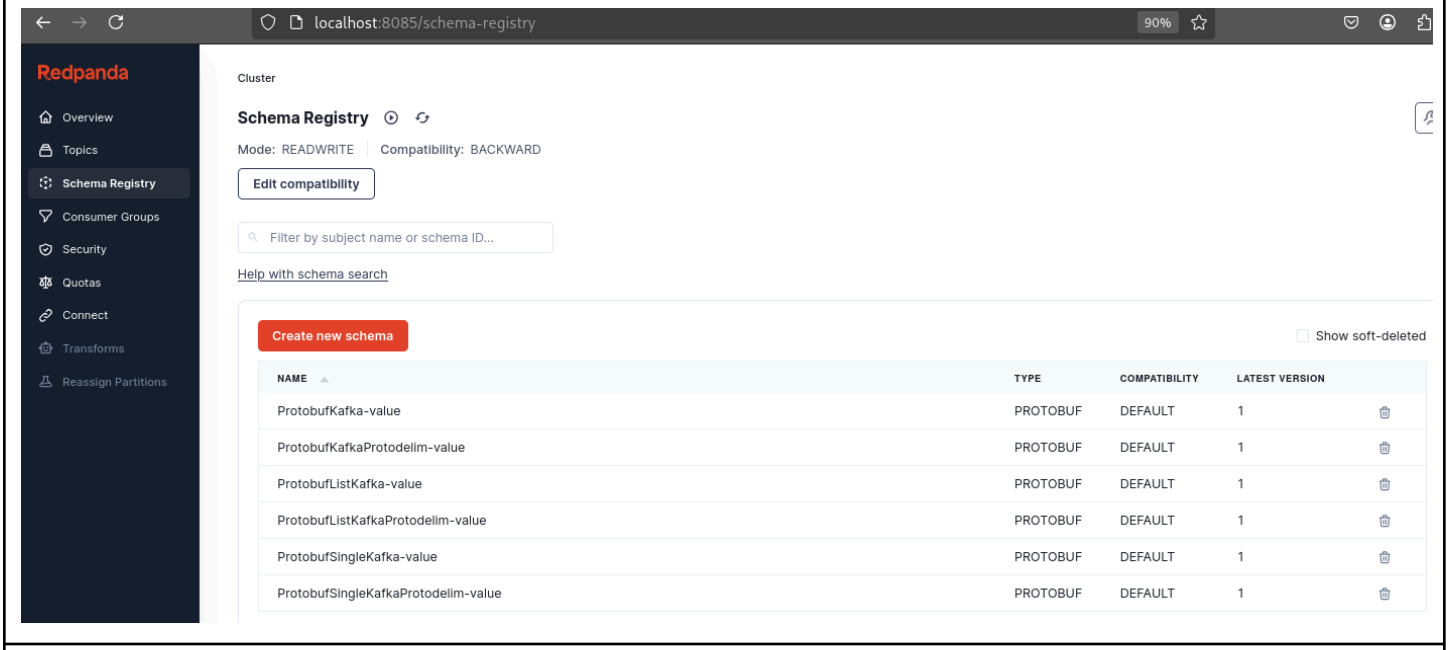
Please note that if you did want to run the 045_recreate_gdp_tables.sh automagically on startup, move the script down one level in the directory hierarchy, and it will be run when clickhouse restarts. I'm not doing that automatically currently, until it's all working.

Schema Registry

The gdp repo registers protobufs with the Redpanda schema registry, which allows the Redpanda pipeline to deserialize the protobufs for operational visibility.

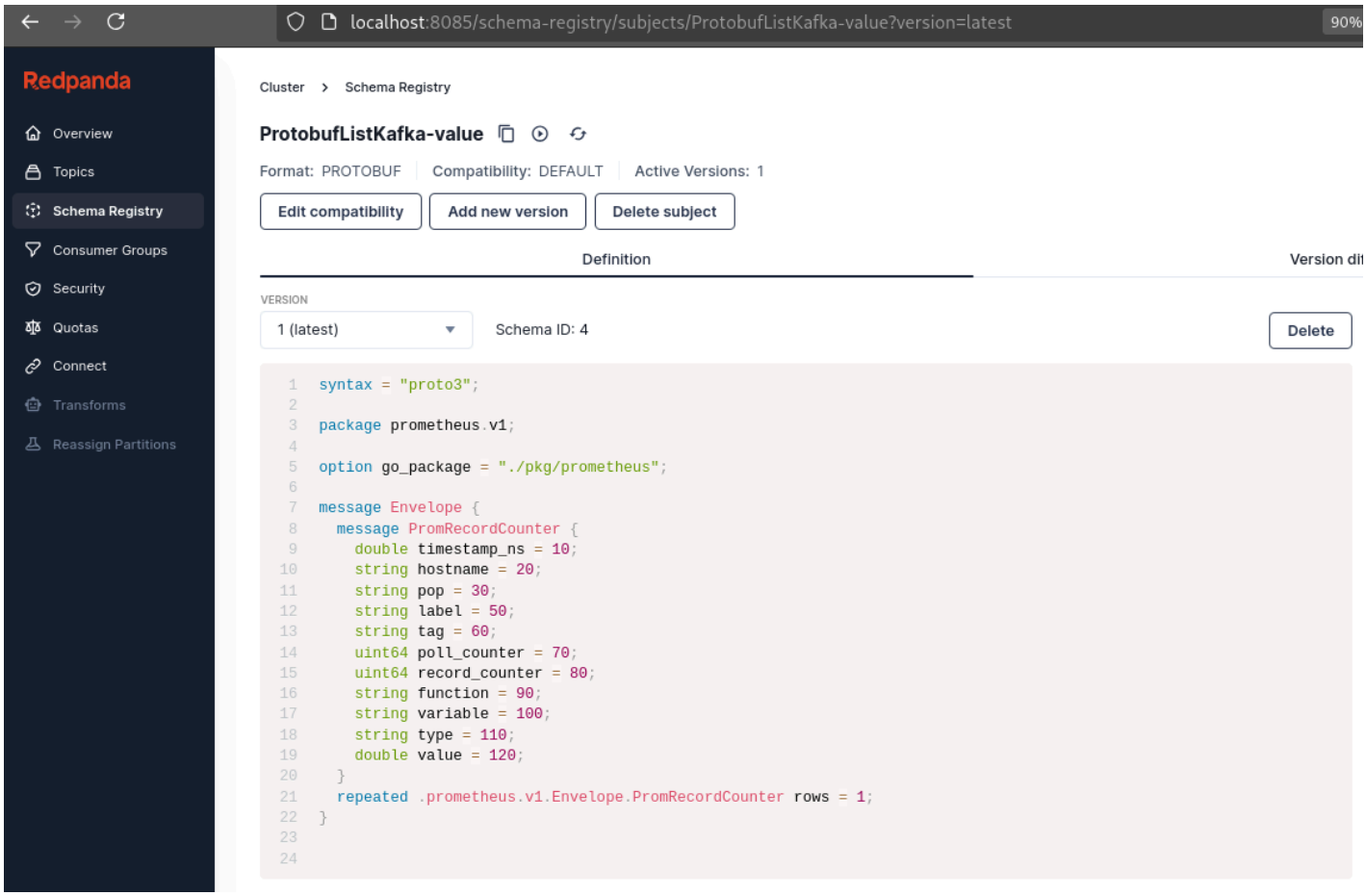
Please note that this code does not bother using franz-go's schema registry helpers, instead it just uses the Redpanda REST interface using the standard go http client library. The franz-go one does work, but the HTTP REST implementation is arguably more readable for go users. I'm currently not using franz-go for protobuf consumption, but suspect that if I was, the franz-go schema registry helpers would improve the ergonomics.
https://github.com/randomizedcoder/gdp/blob/main/pkg/gdp/schema_registry.go#L112

Here are some screenshots of registered protobufs. Here we see an example of x2 registered schemas.



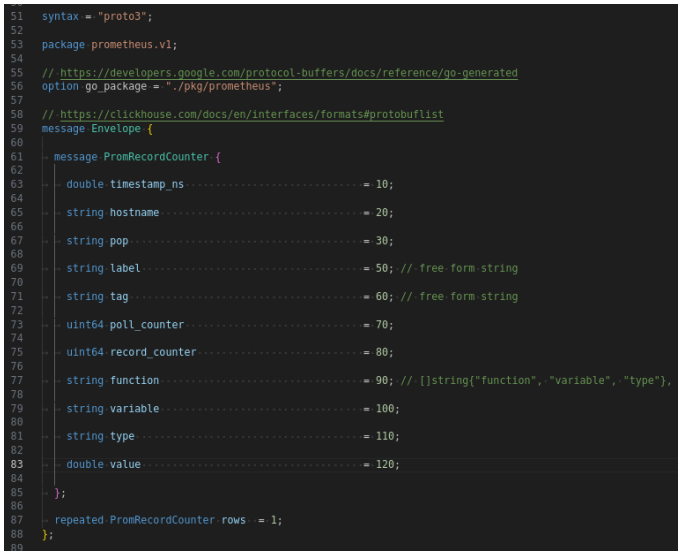
Example of multiple protobuf registered within the Redpanda protobuf schema registry

Here we see the actual protobuf registered into the schema. This view allows you to see the Schema ID, which is critical for streaming so the protobufs are mapped correctly.



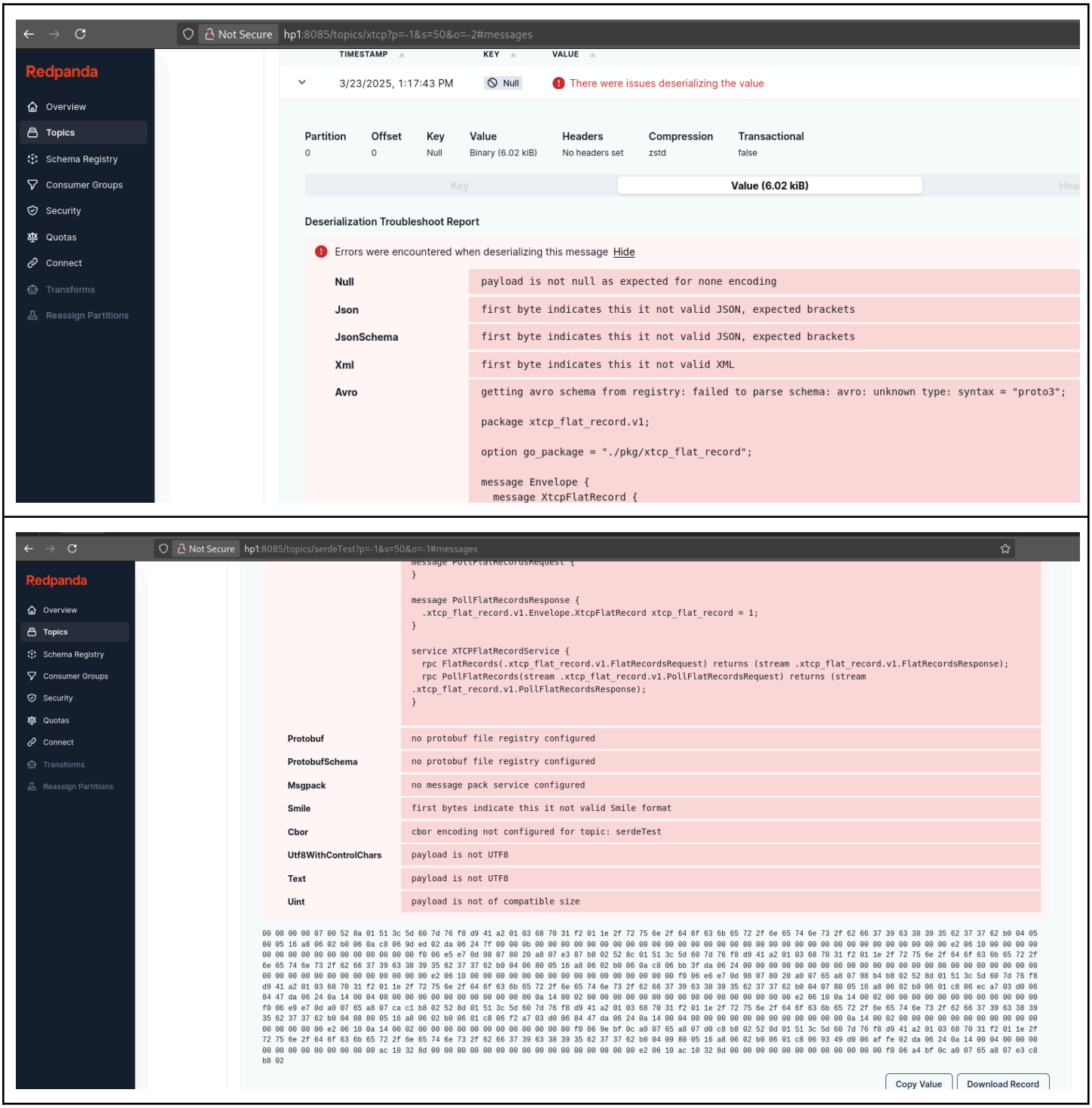
Example of a protobuf registered within the Redpanda protobuf schema registry

One nice thing about the protobuf schema registry, is that you will know if Redpanda is happy about your protobuf if you see it in the GUI and you can see that it has “interpreted” the protobuf. For example, below you can see the original protobuf that was inserted into the registry, we can see that the line numbers have all changed and the line “repeated PromRecordCounter rows = 1;” has been expanded to row 21 above. Similarly, the comments are removed.



Snippet of the original proto

Please note that when Redpanda can't recognise the protobufs in a topic, the GUI is a little misleading. For some reason, the Redpanda GUI apparently shows you that it's trying to decode the data as *all* of the data types, which is confusing. I'd recommend mostly ignoring the errors, because although it looks like Redpanda is trying to use "avro", for example, it is apparently Redpanda is NOT trying to decode "avro".

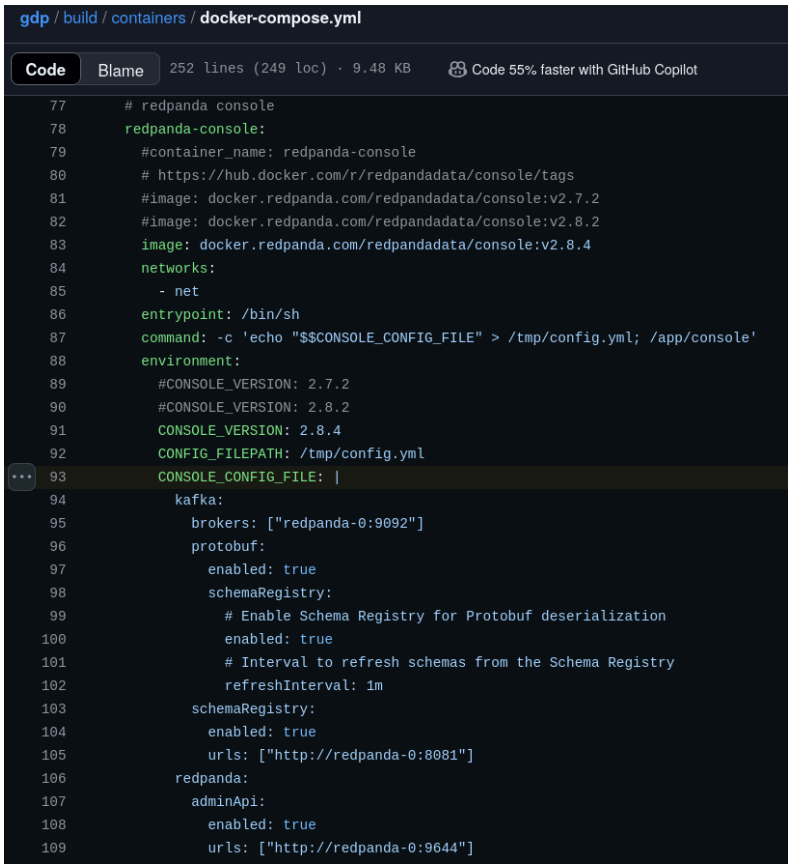


Please also note that Redpanda demonstration containers in their getting started guide do NOT have the protobuf schema registry support enabled by default. Unfortunately there is very little to indicate that the feature is disabled, other than it

not working. Specifically, if the protobuf schema registry is not enabled, the schema registry APIs still accept registering a protobuf. This is a bug if you ask me. If the feature is disabled, it should not accept registration.

The redpanda documentation for this is here:
https://docs.redpanda.com/current/console/config/deserialization/#sr-protobuf

In this repo, you can see in the docker compose that the protobuf schema registry is enabled. Here is also an example docker-compose.yml configuration showing the enabled protobuf schema.



gdp / build / containers / **docker-compose.yml**

Code | Blame | 252 lines (249 loc) · 9.48 KB | Code 55% faster with GitHub Copilot

```
77      # redpanda console
78      redpanda-console:
79        #container_name: redpanda-console
80        # https://hub.docker.com/r/redpandadata/console/tags
81        #image: docker.redpanda.com/redpandadata/console:v2.7.2
82        #image: docker.redpanda.com/redpandadata/console:v2.8.2
83        image: docker.redpanda.com/redpandadata/console:v2.8.4
84        networks:
85          - net
86        entrypoint: /bin/sh
87        command: -c 'echo "$$CONSOLE_CONFIG_FILE" > /tmp/config.yml; /app/console'
88        environment:
89          #CONSOLE_VERSION: 2.7.2
90          #CONSOLE_VERSION: 2.8.2
91          CONSOLE_VERSION: 2.8.4
92          CONFIG_FILEPATH: /tmp/config.yml
93          CONSOLE_CONFIG_FILE: |
94            kafka:
95              brokers: ["redpanda-0:9092"]
96              protobuf:
97                enabled: true
98                schemaRegistry:
99                  # Enable Schema Registry for Protobuf deserialization
100                 enabled: true
101                 # Interval to refresh schemas from the Schema Registry
102                 refreshInterval: 1m
103             schemaRegistry:
104               enabled: true
105               urls: ["http://redpanda-0:8081"]
106           redpanda:
107             adminApi:
108               enabled: true
109               urls: ["http://redpanda-0:9644"]
```

```
[das@hp1:~/xtcp2]$ cat ~/config.yml
kafka:
  brokers: ["redpanda-0:9092"]
  protobuf:
    enabled: true
    schemaRegistry:
      # Enable Schema Registry for Protobuf deserialization.
      enabled: true
      # Interval to refresh schemas from the Schema Registry.
      refreshInterval: 5m
  schemaRegistry:
    enabled: true
    urls: ["http://redpanda-0:8081"]
redpanda:
  adminApi:
    enabled: true
    urls: ["http://redpanda-0:9644"]
```

https://github.com/randomizedcoder/gdp/blob/main/build/containers/docker-compose.yml#L93

The other thing to mention, is that even if the protobufs are registered in the schema, looking in the GUI it may not look like it's working for some time. I suspect this is to do with the "refreshInterval", which I cranked down to 1 minute in the

repo. E.g. Several times I was pulling my hair out, so I went to get a cup of tea, and then found the schema working when I got back. (To nuke the redpanda storage and redeploy just gdp, use “make restart_gdp”). (I suppose, instead of polling, if Redpanda had an event stream system, whenever the protobuf schema registry changed, a notification event could be sent. At the end of the day, most protobuf definitions don’t change very frequently, so this isn’t a big deal.)

Kafka Header

The kafka message header allows the kafka/redpanda to lookup the schemaID in the registry and then deserialize the messages. This is very helpful for diagnostic purposes, but also it’s possible to apply KSQL to filter messages

The documentation for the kafka header is here:
https://docs.confluent.io/platform/current/schema-registry/fundamentals/serdes-develop/index.html#wire-format

Docs repeated here, just to capture them at this point in time:

Wire format

In most cases, you can use the serializers and formatter directly and not worry about the details of how messages are mapped to bytes. However, if you're working with a language that Confluent has not developed serializers for, or simply want a deeper understanding of how the Confluent Platform works, here is more detail on how data is mapped to low-level bytes.

The wire format currently has only a couple of components:

Bytes	Area	Description
0	Magic Byte	Confluent serialization format version number; currently always `0`.
1-4	Schema ID	4-byte schema ID as returned by Schema Registry.
5-...	Data	Serialized data for the specified schema format (for example, binary encoding for Avro or Protocol Buffers). The only exception is raw bytes, which will be written directly without any special encoding.

Important

- All components are encoded with big-endian ordering; that is, standard network byte order.
- The wire format applies to both Kafka message keys and message values.

The Protobuf serialization format appends a list of message indexes after the `magic-byte` and `schema-id`. So, the Protobuf serialization format is:

```
magic-byte, schema-id, message-indexes, protobuf-payload
```

where `message-indexes` is an array of indexes that corresponds to the message type (which may be nested). A single Schema Registry Protobuf entry may contain multiple Protobuf messages, some of which may have nested messages. The role of `message-indexes` is to identify which Protobuf message in the Schema Registry entry to use. For example, given a Schema Registry entry with the following definition:

```
package test.package;

message MessageA {
    message Message B {
        message Message C {
        ...
        }
    }
    message Message D {
    ...
    }
    message Message E {
        message Message F {
        ...
        }
        message Message G {
        ...
        }
    ...
    }
...
}
message MessageH {
    message MessageI {
    ...
    }
}
```

The array `[1, 0]` is (reading the array backwards) the first nested message type of the second top-level message type, corresponding to `test.package.MessageH.MessageI`. Similarly `[0, 2, 1]` is the second message type of the third message type of the first top-level message type corresponding to `test.package.MessageA.MessageE.MessageG`.

The message indexes are encoded as `int` using variable-length zig-zag encoding, the same as Avro (see Binary encoding in the Avro specification), prefixed by the length of the array (which is also variable length, Zigzag encoded). So the above array `[1, 0]` is encoded as the variable length ints `2,1,0` where the first `2` is the length. Also since most of the time the actual message type will be just the first message type (which is the array `[0]`), which would normally be encoded as `1,0` (`1` for length), this special case is optimized to just `0`. So in the most common case of the first message type being used, a single `0` is encoded as the message-indexes.

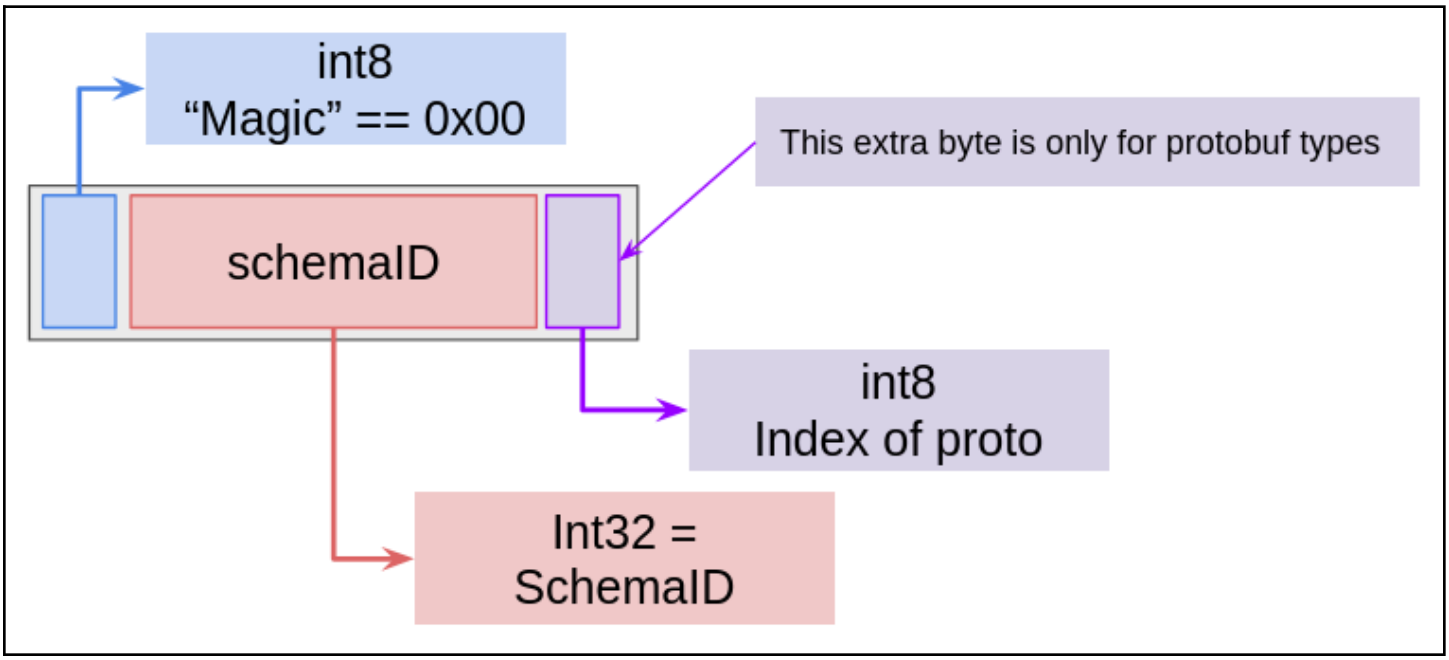
Messages with null keys or values will pass broker-side schema ID validation, as described in the sections on Confluent Cloud and Confluent Platform:

- Using Broker-Side Schema ID Validation on Confluent Cloud
- Validate Broker-side Schemas IDs in Confluent Platform

If anybody from Confluent reads this, my friendly feedback is that the description of the message-index stuff could probably be simplified to improve ease of comprehension .E.g. “(reading the array backwards) the first nested message type of the second top-level message type”. Including a couple of simple examples might help also.

Kafka Header - Summary

The header is actually really simple, and essentially you need to just populate the schemaID. I’ve only used protobufs with the index zero (0) for now.



The go code to populate this is pretty straight forward

```
func (g *GDP) marshal(proto any, mc *gdp_config.MarshalConfig) (buf *[]byte) {

	buf = g.destBytesPool.Get().(*[]byte)

	if mc.KafkaHeader {
	// Add the Confluent header for protobuf, which is length 6
	//
https://docs.confluent.io/platform/current/schema-registry/fundamentals/serdes-develop/index.html#
wire-format
	*buf = (*buf)[:KafkaHeaderSizeCst]
	(*buf)[0] = 0x00                                        // Magic byte
	binary.BigEndian.PutUint32((*buf)[1:], mc.SchemaID)    // Sc
	(*buf)[5] = 0x00                                        // the first message
```

https://github.com/randomizedcoder/gdp/blob/main/pkg/gdp/marshallers.go#L40

The franz-go code can also do this for you, and it works. Within gdp

Kafka Header - Redpanda code

When I couldn’t get redpanda to decode the protobufs, before Redpanda kindly pointed out the feature was disabled, I ended up looking at the source to see what’s up. The source clearly shows how to decode the header, but it’s interesting to see the comments… I don’t know what happens if you send a protobuf message that starts with zero (0), but I bet it’s not great. I wonder why Kafka hasn’t introduced a new version with an actual header? …

```
        // NOTE: Kafka's serialization protocol relies on a magic byte to
        // indicate if we have a schema. This has room for false positives, and
        // we can't say for sure if an error is the result of the record not
        // having a schema. Just translate to binary.
        auto res = get_value_schema_id(*b);
        if (res.has_error()) {
        vlog(datalake_log.trace, "Error parsing schema ID: {}", res.error());
```

https://github.com/redpanda-data/redpanda/blob/93edacb1d4c802c47d239cf7bbdc1660c869bd01/src/v/datalake/record_schema_resolver.cc#L298C1-L305C9

```
get_schema_id_result get_value_schema_id(iobuf& buf) noexcept {
        // Messages that use the schema registry have a 5-byte prefix:
        // offset 0: magic byte with value 0
        // offsets 1-4: schema ID as big-endian integer
        const uint8_t schema_registry_magic_byte = 0;
```

https://github.com/redpanda-data/redpanda/blob/dev/src/v/datalake/schema_registry.cc#L20

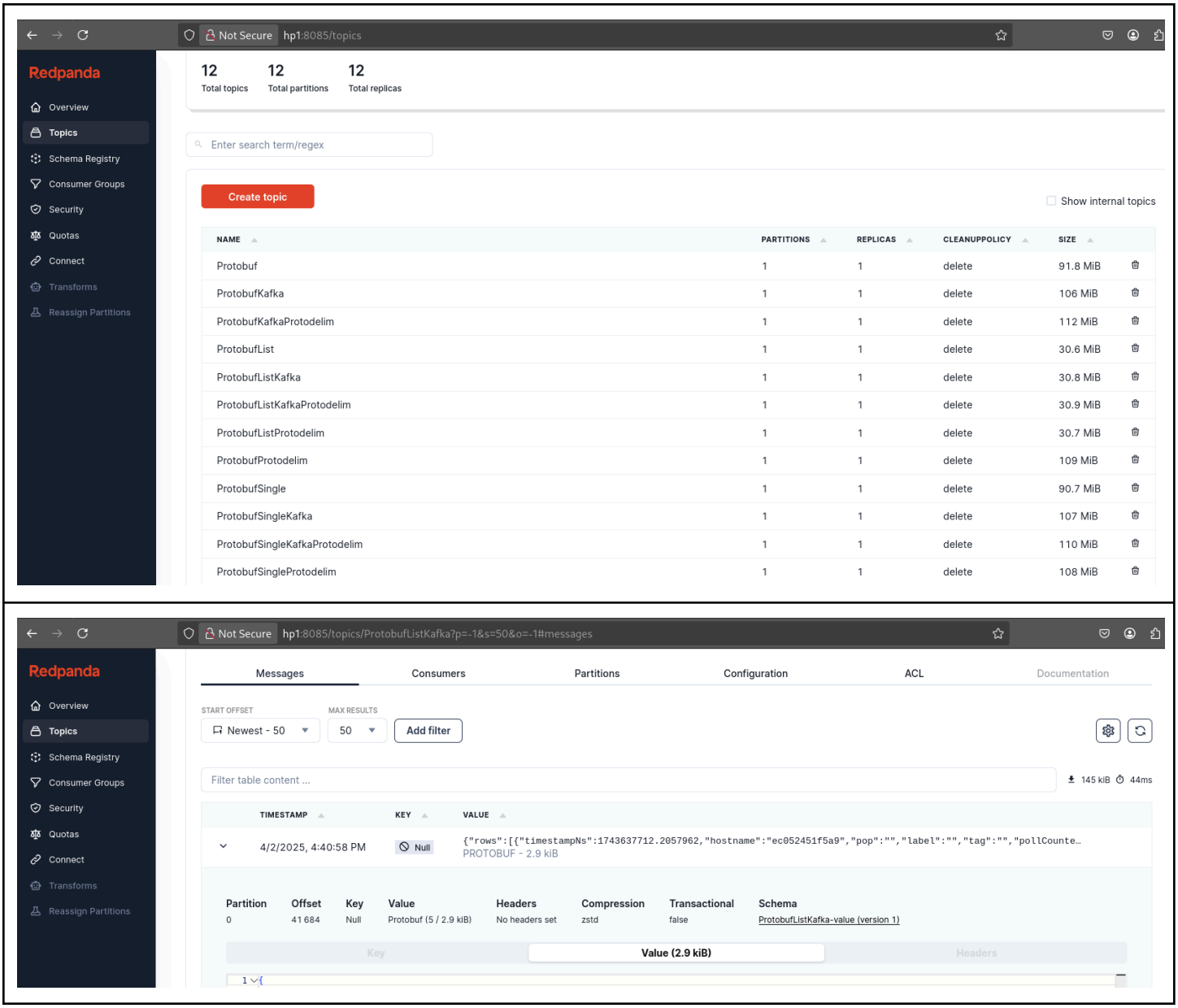
Redpanda Schema Registry Summary

Once you work out how to enable the Redpanda schema registry, it works pretty well! 🚀

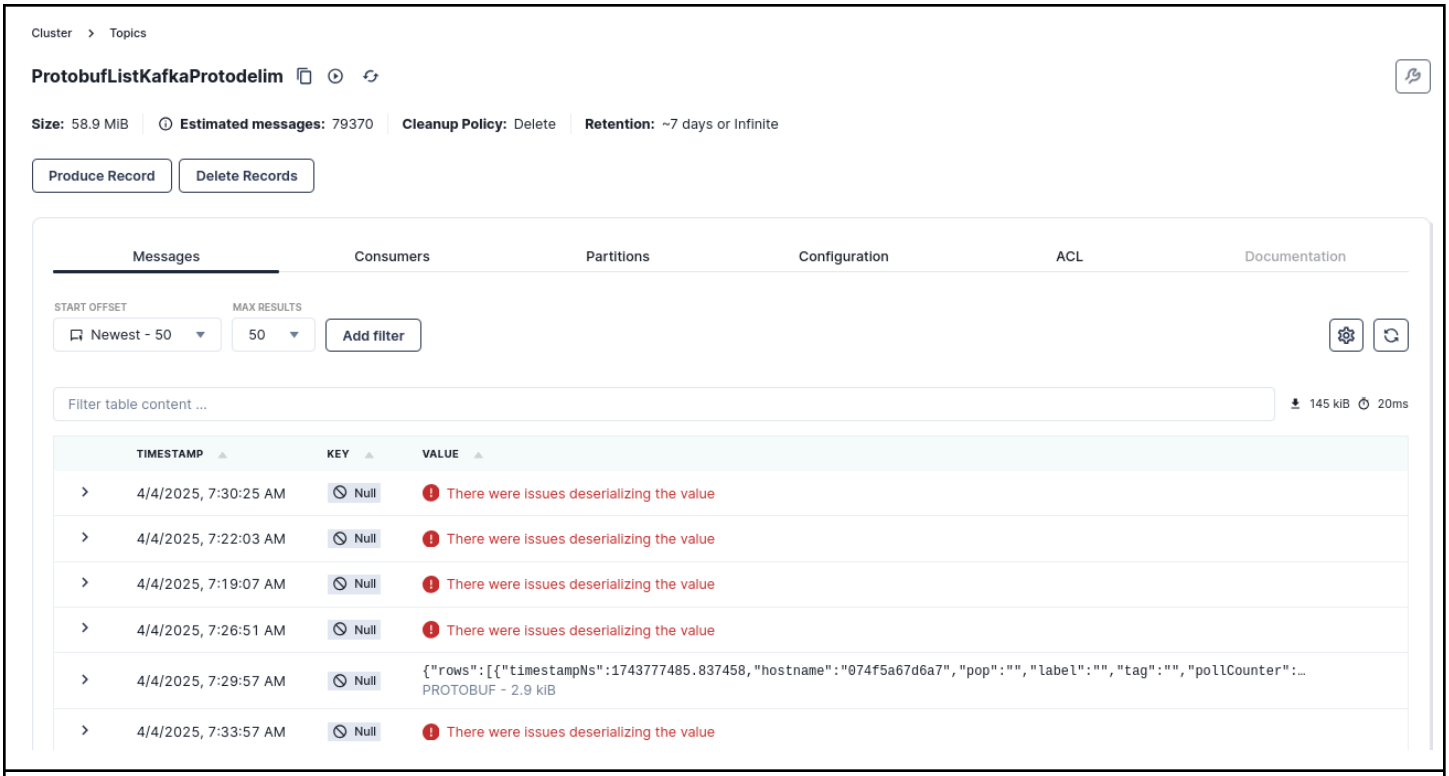
ProtobufList - Envelope efficiency

An interesting observation about using the protobufList Envelope type, compared to the simple protobuf is the significant space saving seen in Redpanda. Here's a screenshot showing the size of the topics after gdb has been running for ~24 hours. We see most of the topics are ~100MB, while the protobufList topics are much closer to ~30MB. I suspect this is because of zstd compression applied across the Envelope getting much more opportunity for de-duplication. E.g. The value of hostname, and many of the function,variable,and type, fields are going to be duplicated across multiple rows.

This strongly supports the argument for using the ProtobufList type.


Not Secure
hp1:8085/topics
Redpanda
Overview
Topics
Schema Registry
Consumer Groups
Security
Quotas
Connect
Transforms
Reassign Partitions
12
Total topics
12
Total partitions
12
Total replicas
Enter search term/regex
Create topic
Show internal topics
NAME
PARTITIONS
REPLICAS
CLEANUPPOLICY
SIZE
Protobuf 1 1 delete 91.8 MiB
ProtobufKafka 1 1 delete 106 MiB
ProtobufKafkaProtodelim 1 1 delete 112 MiB
ProtobufList 1 1 delete 30.6 MiB
ProtobufListKafka 1 1 delete 30.8 MiB
ProtobufListKafkaProtodelim 1 1 delete 30.9 MiB
ProtobufListProtodelim 1 1 delete 30.7 MiB
ProtobufProtodelim 1 1 delete 109 MiB
ProtobufSingle 1 1 delete 90.7 MiB
ProtobufSingleKafka 1 1 delete 107 MiB
ProtobufSingleKafkaProtodelim 1 1 delete 110 MiB
ProtobufSingleProtodelim 1 1 delete 108 MiB

Not Secure
hp1:8085/topics/ProtobufListKafka?p=-1&s=50&o=-1#messages
Redpanda
Overview
Topics
Schema Registry
Consumer Groups
Security
Quotas
Connect
Transforms
Reassign Partitions
Messages
Consumers
Partitions
Configuration
ACL
Documentation
START OFFSET
Newest - 50
MAX RESULTS
50
Add filter
Filter table content ...
145 kiB
44ms
TIMESTAMP
KEY
VALUE
4/2/2025, 4:40:58 PM
Null
{"rows":[{"timestampNs":1743637712.2057962,"hostname":"ec052451f5a9","pop":"","label":"","tag":"","pollCounte…
PROTOBUF - 2.9 kiB
Partition
0
Offset
41 684
Key
Null
Value
Protobuf (5 / 2.9 kiB)
Headers
No headers set
Compression
zstd
Transactional
false
Schema
ProtobufListKafka-value (version 1)
Key
Value (2.9 kiB)
Headers
1 {



Not sure how Redpanda was able to interpret one row. This shouldn't work, because Protodelim has the length varint to allow Clickhouse to process the record.

GDP - Sync.Pool

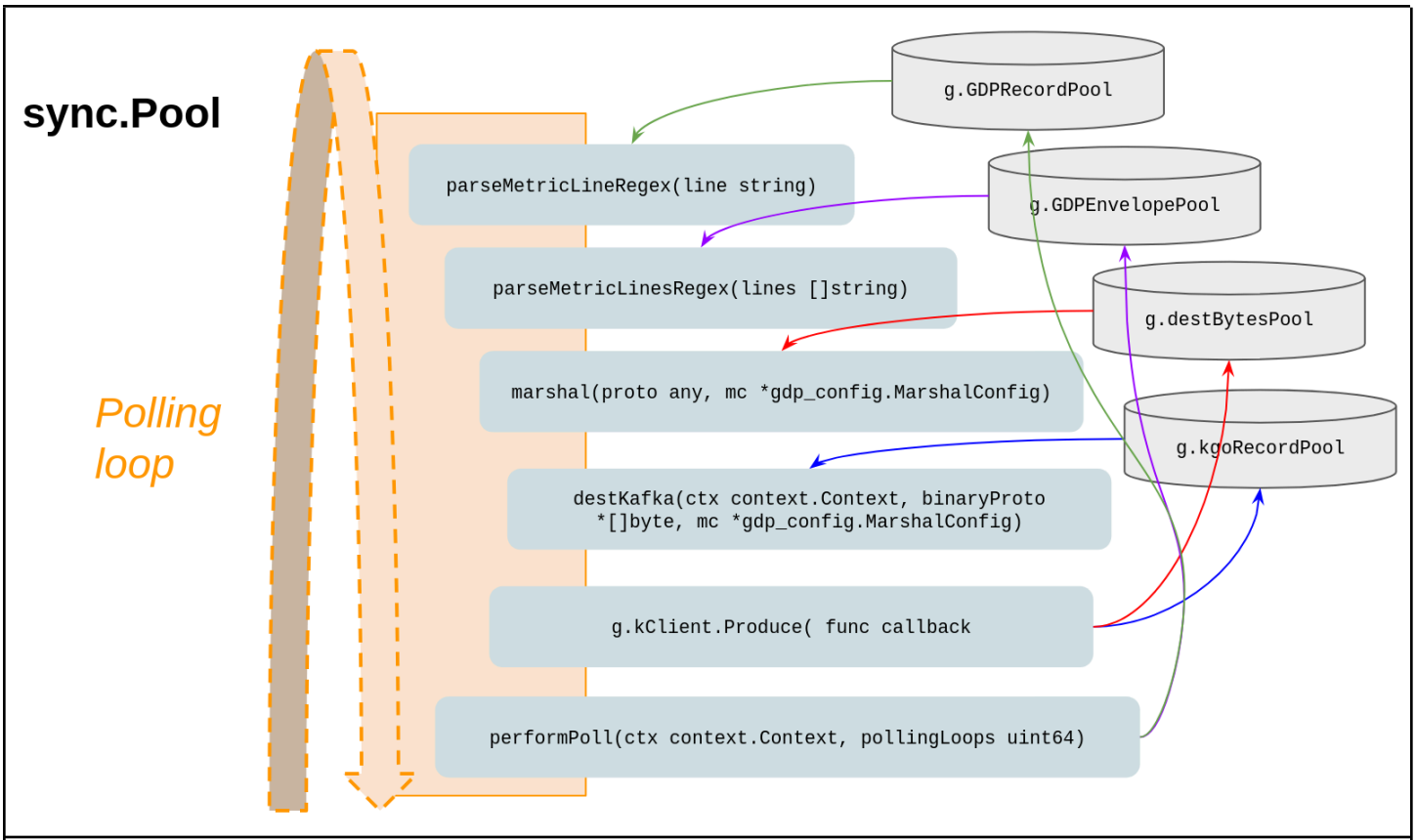
Given that gdp is just a demo, and it's doing ridiculous things like writing protobufs many different ways, there's really no need to use sync.pool. However, the gdp code is a cut down version of other code xtcp, and it already had all the sync.Pool code, so I just reused it. However, given this is a demo, it's probably worth mentioning how the sync.pools work.

Sync.pools mean that golang won't necessarily garbage collect (GC) the objects in the pools, allowing any allocations to be reused. This means that for the objects that are being briefly used, we can reduce the pressure on the memory allocations and on the GC, by using the sync.Pool.

Doesn't this mean that you have to essentially manage memory? Yes. It does seem a little crazy to use a language/runtime that does GC for you and then not use it, but for performance sensitive code it makes sense.

(For those interested, here's a nice video about Prometheus memory optimizations: https://youtu.be/rRtihWOcaLI?si=5u6itVx-dg8UV22D)

Anyway, once you are managing the pools, you need to be careful to .Get(and .Put(in the correct place, or things will get nasty. Here's a little diagram of the way the pools work in gdp.



GDP sync.pools overview

The careful observer will note that the performPoll actually resets each row before returning them to the pool. This is probably not strictly required, because we're dealing with slices, and so resetting the position pointer effectively resets them, but to be really really sure the code was correct, for this demo, gdp is doing some extra work. (See also: https://go.dev/blog/slices-intro). Link to the resetting code here: https://github.com/randomizedcoder/gdp/blob/main/pkg/gdp/poller.go#L185

GDP Status - What actually works in the repo?

ProtobufSingle = Works!

ProtobufSingle is really great! It means you can just call Marshal on the proto and send it to Kafka.

Cons:

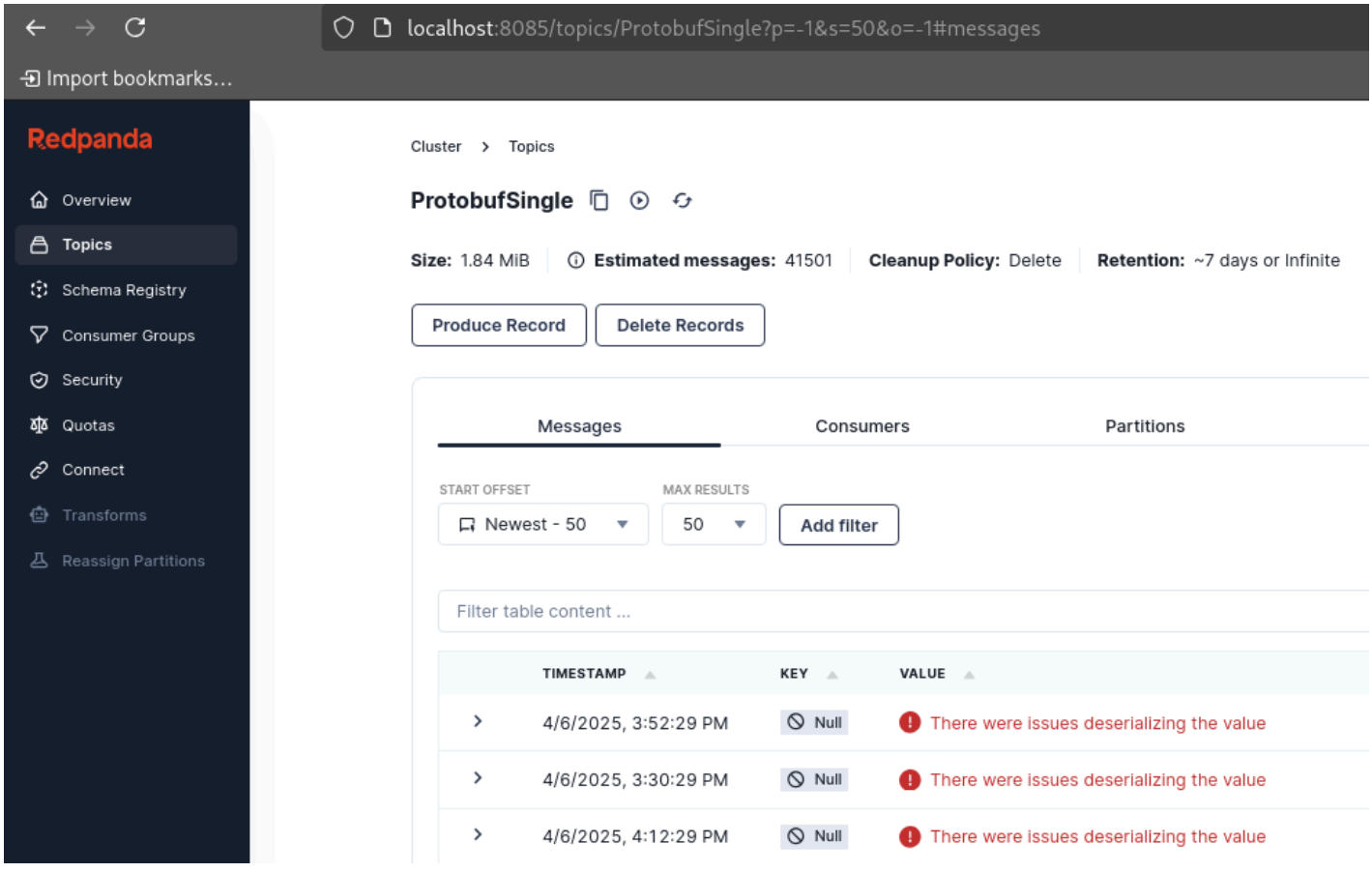
- Redpanda won't be able to read this type, because it has no Kafka header 🙁
- Clickhouse performance won't be great, because we want to batch

Pros:

- It works!

Show me!

Redpanda can't decode this type, because we don't have the kafka header, but at least we can see the messages:



ProtobufSingle with no kafka header can't be decoded by Redpanda (this is expected)

To demonstrate the ingestion into Clickhouse we need to execute the .sql for ProtobufSingle.
https://github.com/randomizedcoder/gdp/blob/main/build/containers/clickhouse/initdb.d/sql/gdp/ProtobufSingle.sql
https://github.com/randomizedcoder/gdp/blob/main/build/containers/clickhouse/initdb.d/sql/gdp/ProtobufSingle_kafka.sql
https://github.com/randomizedcoder/gdp/blob/main/build/containers/clickhouse/initdb.d/sql/gdp/ProtobufSingle_mv.sql

Create the tables

```
788427f2113d :) use gdp

USE gdp

Query id: db58ca8d-5c20-4539-96df-43548f6839e8

Ok.

0 rows in set. Elapsed: 0.002 sec.

788427f2113d :) show tables

SHOW TABLES

Query id: afdd9d6b-cc8a-411a-bd11-9a71e8c816db

Ok.

0 rows in set. Elapsed: 0.002 sec.

788427f2113d :) --
-- gdp.ProtobufSingle.sql
--
DROP TABLE IF EXISTS gdp.ProtobufSingle;

CREATE TABLE IF NOT EXISTS gdp.ProtobufSingle (
  Timestamp_Ns DateTime64(9,'UTC') CODEC(DoubleDelta, LZ4),
  Hostname LowCardinality(String) CODEC(LZ4),
  Pop LowCardinality(String) CODEC(LZ4),
  Label LowCardinality(String) CODEC(LZ4),
  Tag LowCardinality(String) CODEC(LZ4),
  Poll_Counter UInt64 CODEC(DoubleDelta, LZ4),
  Record_Counter UInt64 CODEC(DoubleDelta, LZ4),
  Function LowCardinality(String) CODEC(LZ4),
  Variable LowCardinality(String) CODEC(LZ4),
  Type LowCardinality(String) CODEC(LZ4),
  Value Float64,
)
ENGINE = MergeTree()
PARTITION BY toYYYYMM(Timestamp_Ns)
ORDER BY (Timestamp_Ns, Hostname, Pop, Label, Tag, Poll_Counter, Record_Counter)
TTL toDateTime(Timestamp_Ns) + INTERVAL 14 DAY;

-- Note that ORDER BY clause implicitly specifies a primary key

-- SHOW CREATE TABLE gdp.ProtobufSingle;
-- SELECT * FROM gdp.ProtobufSingle LIMIT 20;

-- https://clickhouse.com/docs/guides/developer/ttl
-- https://clickhouse.com/docs/sql-reference/statements/alter/ttl
-- https://clickhouse.com/docs/engines/table-engines/mergetree-family/mergetree#table_engine-mergetree-ttl
-- https://clickhouse.com/docs/sql-reference/functions/type-conversion-functions#todatetime

-- end



DROP TABLE IF EXISTS gdp.ProtobufSingle

Query id: 37fe524f-adac-40b9-a2d9-7dbfebd410c5

Ok.

0 rows in set. Elapsed: 0.001 sec.


CREATE TABLE IF NOT EXISTS gdp.ProtobufSingle
(
          `Timestamp_Ns` DateTime64(9, 'UTC') CODEC(DoubleDelta, LZ4),
          `Hostname` LowCardinality(String) CODEC(LZ4),
          `Pop` LowCardinality(String) CODEC(LZ4),
          `Label` LowCardinality(String) CODEC(LZ4),
          `Tag` LowCardinality(String) CODEC(LZ4),
          `Poll_Counter` UInt64 CODEC(DoubleDelta, LZ4),
          `Record_Counter` UInt64 CODEC(DoubleDelta, LZ4),
          `Function` LowCardinality(String) CODEC(LZ4),
          `Variable` LowCardinality(String) CODEC(LZ4),
          `Type` LowCardinality(String) CODEC(LZ4),
          `Value` Float64
)
ENGINE = MergeTree
PARTITION BY toYYYYMM(Timestamp_Ns)
ORDER BY (Timestamp_Ns, Hostname, Pop, Label, Tag, Poll_Counter, Record_Counter)
TTL toDateTime(Timestamp_Ns) + toIntervalDay(14)

Query id: 164ce78e-8f54-4574-a457-5b266af0d7ac

Ok.

0 rows in set. Elapsed: 0.007 sec.

788427f2113d :) --
-- gdp.ProtobufSingle_kafka.sql
--
DROP TABLE IF EXISTS gdp.ProtobufSingle_kafka;

CREATE TABLE IF NOT EXISTS gdp.ProtobufSingle_kafka (
  Timestamp_Ns DateTime64(9,'UTC') CODEC(DoubleDelta, LZ4),
  Hostname LowCardinality(String) CODEC(LZ4),
  Pop LowCardinality(String) CODEC(LZ4),
  Label LowCardinality(String) CODEC(LZ4),
  Tag LowCardinality(String) CODEC(LZ4),
```

```
  Poll_Counter UInt64 CODEC(DoubleDelta, LZ4),
  Record_Counter UInt64 CODEC(DoubleDelta, LZ4),
  Function LowCardinality(String) CODEC(LZ4),
  Variable LowCardinality(String) CODEC(LZ4),
  Type LowCardinality(String) CODEC(LZ4),
  Value Float64,
)
ENGINE = Kafka SETTINGS
  kafka_broker_list = 'redpanda-0:9092',
  kafka_topic_list = 'ProtobufSingle',
  kafka_schema = 'prometheus_protolist.proto:PromRecordCounter',
  kafka_max_rows_per_message = 10000,
  kafka_num_consumers = 1,
  kafka_thread_per_consumer = 0,
  kafka_group_name = 'ProtobufSingle',
  kafka_skip_broken_messages = 1,
  kafka_handle_error_mode = 'stream',
  kafka_format = 'ProtobufSingle';

-- SHOW CREATE TABLE gdp.ProtobufSingle_kafka;
-- SELECT * FROM system.kafka_consumers FORMAT Vertical;
-- DETACH TABLE gdp.ProtobufSingle_kafka;
-- SELECT * FROM gdp.ProtobufSingle_kafka LIMIT 20;

-- https://clickhouse.com/docs/integrations/kafka/kafka-table-engine
-- https://clickhouse.com/docs/engines/table-engines/integrations/kafka#creating-a-table
-- kafka_format last! = https://github.com/ClickHouse/ClickHouse/issues/37895

-- end



DROP TABLE IF EXISTS gdp.ProtobufSingle_kafka

Query id: 3333daa5-df0a-46df-8e78-31f2d324dd75

Ok.

0 rows in set. Elapsed: 0.001 sec.


CREATE TABLE IF NOT EXISTS gdp.ProtobufSingle_kafka
(
          `Timestamp_Ns` DateTime64(9, 'UTC') CODEC(DoubleDelta, LZ4),
          `Hostname` LowCardinality(String) CODEC(LZ4),
          `Pop` LowCardinality(String) CODEC(LZ4),
          `Label` LowCardinality(String) CODEC(LZ4),
          `Tag` LowCardinality(String) CODEC(LZ4),
          `Poll_Counter` UInt64 CODEC(DoubleDelta, LZ4),
          `Record_Counter` UInt64 CODEC(DoubleDelta, LZ4),
          `Function` LowCardinality(String) CODEC(LZ4),
          `Variable` LowCardinality(String) CODEC(LZ4),
          `Type` LowCardinality(String) CODEC(LZ4),
          `Value` Float64
)
ENGINE = Kafka
SETTINGS kafka_broker_list = 'redpanda-0:9092', kafka_topic_list = 'ProtobufSingle', kafka_schema = 'prometheus_protolist.proto:PromRecordCounter',
kafka_max_rows_per_message = 10000, kafka_num_consumers = 1, kafka_thread_per_consumer = 0, kafka_group_name = 'ProtobufSingle',
kafka_skip_broken_messages = 1, kafka_handle_error_mode = 'stream', kafka_format = 'ProtobufSingle'

Query id: 90bc0eaf-3cac-4583-a681-03d60b9be2a1

Ok.

0 rows in set. Elapsed: 0.004 sec.

788427f2113d :) --
-- gdp.ProtobufSingle_mv.sql
--
DROP VIEW IF EXISTS gdp.ProtobufSingle_mv;

CREATE MATERIALIZED VIEW gdp.ProtobufSingle_mv TO gdp.ProtobufSingle
  AS SELECT
          *,
--        toDateTime64(Timestamp_Ns, 9, 'UTC') AS Timestamp_Ns,
--        Hostname,
--        Pop,
--        Label,
--        Tag,
--        Poll_Counter,
--        Record_Counter,
--        Function,
--        Variable,
--        Type,
--        Value,
  FROM gdp.ProtobufSingle_kafka;

-- SHOW CREATE TABLE gdp.ProtobufSingle_mv;

-- https://clickhouse.com/docs/sql-reference/statements/create/view#materialized-view

-- See also:
-- https://github.com/ClickHouse/ClickHouse/blob/master/tests/integration/test_storage_kafka/test_batch_fast.py#L2679
-- https://github.com/ClickHouse/ClickHouse/blob/master/tests/integration/test_storage_kafka/test_batch_slow.py

-- end



DROP VIEW IF EXISTS gdp.ProtobufSingle_mv

Query id: e14314b2-5660-4587-a50c-ec5457c43185

Ok.
```

```
0 rows in set. Elapsed: 0.001 sec.


CREATE MATERIALIZED VIEW gdp.ProtobufSingle_mv TO gdp.ProtobufSingle
AS SELECT *
FROM gdp.ProtobufSingle_kafka

Query id: 9de60407-60b7-4afb-bb33-4ba218625906

Ok.

0 rows in set. Elapsed: 0.005 sec.

788427f2113d :) show tables;

SHOW TABLES

Query id: 5d5d51e7-d3c7-418b-a325-958fa1af9d8a

   ┌─name──────────────────┐
1. │ ProtobufSingle        │
2. │ ProtobufSingle_kafka  │
3. │ ProtobufSingle_mv     │
   └───────────────────────┘

3 rows in set. Elapsed: 0.001 sec.
```

Create ProtobufSingle tables

Check if kafka is working

```
788427f2113d :)  SELECT * FROM system.kafka_consumers FORMAT Vertical;

SELECT *
FROM system.kafka_consumers
FORMAT Vertical

Query id: 95b47843-bb46-479c-8a0d-27c91d8dbe07

Row 1:
──────
database:                    gdp
table:                       ProtobufSingle_kafka
consumer_id:
ClickHouse-788427f2113d-gdp-ProtobufSingle_kafka-70cc3aa2-8d67-43a5-a0a2-04cf0a22c68f
assignments.topic:           ['ProtobufSingle']
assignments.partition_id:   [0]
assignments.current_offset: [41641]
exceptions.time:             []
exceptions.text:             []
last_poll_time:              2025-04-06 23:24:18
num_messages_read:           37576
last_commit_time:            2025-04-06 23:23:32
num_commits:                 2
last_rebalance_time:         1970-01-01 00:00:00
num_rebalance_revocations:  0
num_rebalance_assignments:  1
is_currently_used:           1
last_used:                   2025-04-06 23:24:16.912110
rdkafka_stat:                { "name": "ClickHouse-788427f2113d-gdp-ProtobufSingle_kafka#consumer-1",
"client_id": "ClickHouse-788427f2113d-gdp-ProtobufSingle_kafka", "type": "consumer", "ts":518945016084,
"time":1743981856, "age":84021115, "replyq":0, "msg_cnt":0, "msg_size":0, "msg_max":0, "msg_size_max":0,
"simple_cnt":0, "metadata_cache_cnt":1, "brokers":{ "redpanda-0:9092/0": { "name":"redpanda-0:9092/0",
"nodeid":0, "nodename":"redpanda-0:9092", "source":"configured", "state":"UP", "stateage":84019145,
"outbuf_cnt":0, "outbuf_msg_cnt":0, "waitresp_cnt":1, "waitresp_msg_cnt":0, "tx":177, "txbytes":25423,
"txerrs":0, "txretries":0, "txidle":390100, "req_timeouts":0, "rx":176, "rxbytes":1769090, "rxerrs":0,
"rxcorriderrs":0, "rxpartial":0, "rxidle":390243, "zbuf_grow":0, "buf_grow":0, "wakeups":445,
"connects":1, "disconnects":0, "int_latency": { "min":0, "max":0, "avg":0, "sum":0, "stddev": 0, "p50": 0,
"p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0, "cnt":0 },
"outbuf_latency": { "min":35, "max":58, "avg":46, "sum":281, "stddev": 0, "p50": 0, "p75": 0, "p90": 0,
"p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0, "cnt":6 }, "rtt": { "min":500137,
"max":500230, "avg":500186, "sum":3001119, "stddev": 0, "p50": 0, "p75": 0, "p90": 0, "p95": 0, "p99": 0,
"p99_99": 0, "outofrange": 0, "hdrsize": 0, "cnt":6 }, "throttle": { "min":0, "max":0, "avg":0, "sum":0,
"stddev": 0, "p50": 0, "p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0,
"cnt":0 }, "req": { "Fetch": 173, "ListOffsets": 0, "Metadata": 1, "OffsetCommit": 0, "OffsetFetch": 0,
"FindCoordinator": 2, "JoinGroup": 0, "Heartbeat": 0, "LeaveGroup": 0, "SyncGroup": 0, "SaslHandshake": 0,
"ApiVersion": 1, "SaslAuthenticate": 0, "DescribeCluster": 0, "DescribeProducers": 0, "Unknown-62?": 0,
"DescribeTransactions": 0, "ListTransactions": 0, "Unknown-69?": 0, "Unknown-70?": 0,
"GetTelemetrySubscriptions": 0, "PushTelemetry": 0, "Unknown-73?": 0 }, "toppars":{ "ProtobufSingle-0": {
"topic":"ProtobufSingle", "partition":0} } } , "GroupCoordinator": { "name":"GroupCoordinator",
```

```
"nodeid":0, "nodename":"redpanda-0:9092", "source":"logical", "state":"UP", "stateage":84017950,
"outbuf_cnt":0, "outbuf_msg_cnt":0, "waitresp_cnt":0, "waitresp_msg_cnt":0, "tx":38, "txbytes":6477,
"txerrs":0, "txretries":0, "txidle":2976702, "req_timeouts":0, "rx":38, "rxbytes":1597, "rxerrs":0,
"rxcorriderrs":0, "rxpartial":0, "rxidle":2976558, "zbuf_grow":0, "buf_grow":0, "wakeups":172,
"connects":1, "disconnects":0, "int_latency": { "min":0, "max":0, "avg":0, "sum":0, "stddev": 0, "p50": 0,
"p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0, "cnt":0 },
"outbuf_latency": { "min":53, "max":53, "avg":53, "sum":53, "stddev": 0, "p50": 0, "p75": 0, "p90": 0,
"p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0, "cnt":1 }, "rtt": { "min":144, "max":144,
"avg":144, "sum":144, "stddev": 0, "p50": 0, "p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0,
"outofrange": 0, "hdrsize": 0, "cnt":1 }, "throttle": { "min":0, "max":0, "avg":0, "sum":0, "stddev": 0,
"p50": 0, "p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0, "cnt":0 },
"req": { "Fetch": 0, "ListOffsets": 0, "Metadata": 1, "OffsetCommit": 2, "OffsetFetch": 3,
"FindCoordinator": 0, "JoinGroup": 2, "Heartbeat": 28, "LeaveGroup": 0, "SyncGroup": 1, "SaslHandshake":
0, "ApiVersion": 1, "SaslAuthenticate": 0, "DescribeCluster": 0, "DescribeProducers": 0, "Unknown-62?": 0,
"DescribeTransactions": 0, "ListTransactions": 0, "Unknown-69?": 0, "Unknown-70?": 0,
"GetTelemetrySubscriptions": 0, "PushTelemetry": 0, "Unknown-73?": 0 }, "toppars":{ } } }, "topics":{
"ProtobufSingle": { "topic":"ProtobufSingle", "age":84015, "metadata_age":84017, "batchsize": { "min":0,
"max":0, "avg":0, "sum":0, "stddev": 0, "p50": 0, "p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0,
"outofrange": 0, "hdrsize": 0, "cnt":0 }, "batchcnt": { "min":0, "max":0, "avg":0, "sum":0, "stddev": 0,
"p50": 0, "p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0, "cnt":0 },
"partitions":{ "0": { "partition":0, "broker":0, "leader":0, "desired":true, "unknown":false,
"msgq_cnt":0, "msgq_bytes":0, "xmit_msgq_cnt":0, "xmit_msgq_bytes":0, "fetchq_cnt":0, "fetchq_size":0,
"fetch_state":"active", "query_offset":-1001, "next_offset":41641, "app_offset":41641,
"stored_offset":41641, "stored_leader_epoch":-1, "commited_offset":41641, "committed_offset":41641,
"committed_leader_epoch":-1, "eof_offset":41641, "lo_offset":0, "hi_offset":41641, "ls_offset":41641,
"consumer_lag":0, "consumer_lag_stored":0, "leader_epoch":-1, "txmsgs":0, "txbytes":0, "rxmsgs":37576,
"rxbytes":3002054, "msgs": 37576, "rx_ver_drops": 0, "msgs_inflight": 0, "next_ack_seq": 0,
"next_err_seq": 0, "acked_msgid": 0} , "-1": { "partition":-1, "broker":-1, "leader":-1, "desired":false,
"unknown":false, "msgq_cnt":0, "msgq_bytes":0, "xmit_msgq_cnt":0, "xmit_msgq_bytes":0, "fetchq_cnt":0,
"fetchq_size":0, "fetch_state":"none", "query_offset":-1001, "next_offset":0, "app_offset":-1001,
"stored_offset":-1001, "stored_leader_epoch":-1, "commited_offset":-1001, "committed_offset":-1001,
"committed_leader_epoch":-1, "eof_offset":-1001, "lo_offset":-1001, "hi_offset":-1001, "ls_offset":-1001,
"consumer_lag":-1, "consumer_lag_stored":-1, "leader_epoch":-1, "txmsgs":0, "txbytes":0, "rxmsgs":0,
"rxbytes":0, "msgs": 0, "rx_ver_drops": 0, "msgs_inflight": 0, "next_ack_seq": 0, "next_err_seq": 0,
"acked_msgid": 0} } } } , "cgrp": { "state": "up", "stateage": 84017, "join_state": "steady",
"rebalance_age": 84015, "rebalance_cnt": 1, "rebalance_reason": "Metadata for subscribed topic(s) has
changed", "assignment_size": 1 }, "tx":215, "tx_bytes":31900, "rx":214, "rx_bytes":1770687, "txmsgs":0,
"txmsg_bytes":0, "rxmsgs":37576, "rxmsg_bytes":3002054}

1 row in set. Elapsed: 0.002 sec.
```


Verify Kafka ingestion
https://github.com/randomizedcoder/gdp/blob/main/build/containers/clickhouse/initdb.d/sql/gdp/ProtobufSingle_kafka.sql#L32

You can also verify the Kafka counters here. The good news is that we see no errors! Yay!

```
788427f2113d :) SELECT
         event,
         value,
         description
FROM system.events
WHERE event LIKE '%Kafka%';

SELECT
         event,
         value,
         description
FROM system.events
WHERE event LIKE '%Kafka%'

Query id: 3eb94c64-0144-4be3-a338-abd89dbf5e27


┌─event─────────────────────┬─value─┬─description──────────────────────────────────────────────────────────────────────────────────────────────────────────────┐
1. │ KafkaRebalanceAssignments │     1 │ Number of partition assignments (the final stage of consumer group rebalance)
2. │ KafkaMessagesPolled       │ 37632 │ Number of Kafka messages polled from librdkafka to ClickHouse
3. │ KafkaMessagesRead         │ 37632 │ Number of Kafka messages already processed by ClickHouse
4. │ KafkaRowsRead             │ 37632 │ Number of rows parsed from Kafka messages
5. │ KafkaBackgroundReads      │    29 │ Number of background reads populating materialized views from Kafka since server start
6. │ KafkaCommits              │     3 │ Number of successful commits of consumed offsets to Kafka (normally should be the same as
KafkaBackgroundReads) │
└───────────────────────────┴───────┴──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┘
```

```
6 rows in set. Elapsed: 0.001 sec.
```

Check Kafka system events
https://github.com/randomizedcoder/gdp/blob/main/build/containers/clickhouse/initdb.d/sql/select_system_events_like_kafka.sql

And we can see the table is getting populated. Woot woot! (Dang Clickhouse is fast!)

```
788427f2113d :) SELECT * FROM gdp.ProtobufSingle LIMIT 20;

SELECT *
FROM gdp.ProtobufSingle
LIMIT 20

Query id: e323ef7d-b4ed-48af-aafe-9c173554d7d8


┌────────────────Timestamp_Ns─┬─Hostname─────┬─Pop─┬─Label─┬─Tag─┬─Poll_Counter─┬─Record_Counter─┬─Function─┬─Variable──────────────────────┬─Ty
pe────┬─Value─┐
 1. │ 2025-04-06 01:02:13.607119360 │ 1d93eab32382 │     │       │     │          146 │              0 │ Poller   │ pollingLoops
       │ count │   147 │
 2. │ 2025-04-06 01:02:13.607119360 │ 1d93eab32382 │     │       │     │          146 │              1 │ Poller   │ ticker
       │ count │   147 │
 3. │ 2025-04-06 01:02:13.607119360 │ 1d93eab32382 │     │       │     │          146 │              2 │ Run      │ start
       │ counter │     1 │
 4. │ 2025-04-06 01:02:13.607119360 │ 1d93eab32382 │     │       │     │          146 │              3 │ destKafka │ ProtobufList
       │ count │   146 │
 5. │ 2025-04-06 01:02:13.607119360 │ 1d93eab32382 │     │       │     │          146 │              4 │ destKafka │ ProtobufListKafka
       │ count │   146 │
 6. │ 2025-04-06 01:02:13.607119360 │ 1d93eab32382 │     │       │     │          146 │              5 │ destKafka │
ProtobufListKafkaProtodelim   │ count │   146 │
 7. │ 2025-04-06 01:02:13.607119360 │ 1d93eab32382 │     │       │     │          146 │              6 │ destKafka │
ProtobufListProtodelim        │ count │   146 │
 8. │ 2025-04-06 01:02:13.607119360 │ 1d93eab32382 │     │       │     │          146 │              7 │ destKafka │ ProtobufSingle
       │ count │  4065 │
 9. │ 2025-04-06 01:02:13.607119360 │ 1d93eab32382 │     │       │     │          146 │              8 │ destKafka │ ProtobufSingleKafka
       │ count │  4065 │
10. │ 2025-04-06 01:02:13.607119360 │ 1d93eab32382 │     │       │     │          146 │              9 │ destKafka │
ProtobufSingleKafkaProtodelim │ count │  4065 │
11. │ 2025-04-06 01:02:13.607119360 │ 1d93eab32382 │     │       │     │          146 │             10 │ destKafka │
ProtobufSingleProtodelim      │ count │  4065 │
12. │ 2025-04-06 01:02:13.607119360 │ 1d93eab32382 │     │       │     │          146 │             11 │ destKafka │ start
       │ count │ 16844 │
13. │ 2025-04-06 01:02:13.607119360 │ 1d93eab32382 │     │       │     │          146 │             12 │ destKafka │ ProtobufList
       │ n     │ 329274 │
14. │ 2025-04-06 01:02:13.607119360 │ 1d93eab32382 │     │       │     │          146 │             13 │ destKafka │ ProtobufListKafka
       │ n     │ 330150 │
15. │ 2025-04-06 01:02:13.607119360 │ 1d93eab32382 │     │       │     │          146 │             14 │ destKafka │
ProtobufListKafkaProtodelim   │ n     │ 330442 │
16. │ 2025-04-06 01:02:13.607119360 │ 1d93eab32382 │     │       │     │          146 │             15 │ destKafka │
ProtobufListProtodelim        │ n     │ 329566 │
17. │ 2025-04-06 01:02:13.607119360 │ 1d93eab32382 │     │       │     │          146 │             16 │ destKafka │ ProtobufSingle
       │ n     │ 321144 │
18. │ 2025-04-06 01:02:13.607119360 │ 1d93eab32382 │     │       │     │          146 │             17 │ destKafka │ ProtobufSingleKafka
       │ n     │ 345534 │
19. │ 2025-04-06 01:02:13.607119360 │ 1d93eab32382 │     │       │     │          146 │             18 │ destKafka │
ProtobufSingleKafkaProtodelim │ n     │ 349599 │
20. │ 2025-04-06 01:02:13.607119360 │ 1d93eab32382 │     │       │     │          146 │             19 │ destKafka │
ProtobufSingleProtodelim      │ n     │ 325209 │

└────────────────────────────┴──────────────┴─────┴───────┴─────┴──────────────┴────────────────┴──────────┴──────────────────────────────┴──
────┴───────┘

20 rows in set. Elapsed: 0.003 sec. Processed 16.38 thousand rows, 640.27 KB (6.11 million rows/s., 238.78 MB/s.)
Peak memory usage: 575.58 KiB.
```

Verify records
https://github.com/randomizedcoder/gdp/blob/main/build/containers/clickhouse/initdb.d/sql/gdp/ProtobufSingle.sql#L27

What doesn't work, but probably should work. Maybe this is a bug?

ProtobufListProtoDelim = Does NOT work :(

Just to remember the naming convention, ProtobufListProtoDelim means:

- This is an Envelope protobuf payload. This *should* mean we get pretty good performance, or at least a lot better performance than ProtobufSingle.

- Kafka header is NOT used, so Redpanda can't see it
- Protodelim means that there is a length delimiting var int, before the actual binary payload of the protobuf.

How to demonstrate this

We are going to run the SQL commands from these files
https://github.com/randomizedcoder/gdp/blob/main/build/containers/clickhouse/initdb.d/sql/gdp/ProtobufListProtodelim.sql
https://github.com/randomizedcoder/gdp/blob/main/build/containers/clickhouse/initdb.d/sql/gdp/ProtobufListProtodelim_kafka.sql
https://github.com/randomizedcoder/gdp/blob/main/build/containers/clickhouse/initdb.d/sql/gdp/ProtobufListProtodelim_mv.sql

```
788427f2113d :) show tables

SHOW TABLES

Query id: 5e9f1370-41cf-45f0-a57d-1b0ac670b0e3

Ok.

0 rows in set. Elapsed: 0.001 sec.

788427f2113d :) --
-- gdp.ProtobufListProtodelim.sql
--
DROP TABLE IF EXISTS gdp.ProtobufListProtodelim;

CREATE TABLE IF NOT EXISTS gdp.ProtobufListProtodelim (
  Timestamp_Ns DateTime64(9,'UTC') CODEC(DoubleDelta, LZ4),
  Hostname LowCardinality(String) CODEC(LZ4),
  Pop LowCardinality(String) CODEC(LZ4),
  Label LowCardinality(String) CODEC(LZ4),
  Tag LowCardinality(String) CODEC(LZ4),
  Poll_Counter UInt64 CODEC(DoubleDelta, LZ4),
  Record_Counter UInt64 CODEC(DoubleDelta, LZ4),
  Function LowCardinality(String) CODEC(LZ4),
  Variable LowCardinality(String) CODEC(LZ4),
  Type LowCardinality(String) CODEC(LZ4),
  Value Float64,
)
ENGINE = MergeTree()
PARTITION BY toYYYYMM(Timestamp_Ns)
ORDER BY (Timestamp_Ns, Hostname, Pop, Label, Tag, Poll_Counter, Record_Counter)
TTL toDateTime(Timestamp_Ns) + INTERVAL 14 DAY;

-- Note that ORDER BY clause implicitly specifies a primary key

-- SHOW CREATE TABLE gdp.ProtobufListProtodelim;
-- SELECT * FROM gdp.ProtobufListProtodelim LIMIT 20;

-- https://clickhouse.com/docs/guides/developer/ttl
-- https://clickhouse.com/docs/sql-reference/statements/alter/ttl
-- https://clickhouse.com/docs/engines/table-engines/mergetree-family/mergetree#table_engine-mergetree-ttl
-- https://clickhouse.com/docs/sql-reference/functions/type-conversion-functions#todatetime

-- end



DROP TABLE IF EXISTS gdp.ProtobufListProtodelim

Query id: d90e5de3-9079-430d-a03a-be5f952832f2

Ok.

0 rows in set. Elapsed: 0.001 sec.


CREATE TABLE IF NOT EXISTS gdp.ProtobufListProtodelim
(
          `Timestamp_Ns` DateTime64(9, 'UTC') CODEC(DoubleDelta, LZ4),
          `Hostname` LowCardinality(String) CODEC(LZ4),
          `Pop` LowCardinality(String) CODEC(LZ4),
          `Label` LowCardinality(String) CODEC(LZ4),
          `Tag` LowCardinality(String) CODEC(LZ4),
          `Poll_Counter` UInt64 CODEC(DoubleDelta, LZ4),
          `Record_Counter` UInt64 CODEC(DoubleDelta, LZ4),
          `Function` LowCardinality(String) CODEC(LZ4),
          `Variable` LowCardinality(String) CODEC(LZ4),
          `Type` LowCardinality(String) CODEC(LZ4),
          `Value` Float64
)
ENGINE = MergeTree
PARTITION BY toYYYYMM(Timestamp_Ns)
ORDER BY (Timestamp_Ns, Hostname, Pop, Label, Tag, Poll_Counter, Record_Counter)
TTL toDateTime(Timestamp_Ns) + toIntervalDay(14)
```

```
Query id: ef805ce1-1263-46c6-92c5-6b28137ab28b

Ok.

0 rows in set. Elapsed: 0.006 sec.

788427f2113d :) --
-- gdp.ProtobufListProtodelim_kafka.sql
--
DROP TABLE IF EXISTS gdp.ProtobufListProtodelim_kafka;

CREATE TABLE IF NOT EXISTS gdp.ProtobufListProtodelim_kafka (
  Timestamp_Ns DateTime64(9,'UTC') CODEC(DoubleDelta, LZ4),
  Hostname LowCardinality(String) CODEC(LZ4),
  Pop LowCardinality(String) CODEC(LZ4),
  Label LowCardinality(String) CODEC(LZ4),
  Tag LowCardinality(String) CODEC(LZ4),
  Poll_Counter UInt64 CODEC(DoubleDelta, LZ4),
  Record_Counter UInt64 CODEC(DoubleDelta, LZ4),
  Function LowCardinality(String) CODEC(LZ4),
  Variable LowCardinality(String) CODEC(LZ4),
  Type LowCardinality(String) CODEC(LZ4),
  Value Float64,
)
ENGINE = Kafka SETTINGS
  kafka_broker_list = 'redpanda-0:9092',
  kafka_topic_list = 'ProtobufListProtodelim',
  kafka_schema = 'prometheus_protolist.proto:PromRecordCounter',
  kafka_max_rows_per_message = 10000,
  kafka_num_consumers = 1,
  kafka_thread_per_consumer = 0,
  kafka_group_name = 'ProtobufListProtodelim',
  kafka_skip_broken_messages = 1,
  kafka_handle_error_mode = 'stream',
  kafka_format = 'ProtobufList';

-- SHOW CREATE TABLE gdp.ProtobufListProtodelim_kafka;
-- SELECT * FROM system.kafka_consumers FORMAT Vertical;
-- DETACH TABLE gdp.ProtobufListProtodelim_kafka;
-- SELECT * FROM gdp.ProtobufListProtodelim_kafka LIMIT 20;

-- https://clickhouse.com/docs/integrations/kafka/kafka-table-engine
-- https://clickhouse.com/docs/engines/table-engines/integrations/kafka#creating-a-table
-- kafka_format last! = https://github.com/ClickHouse/ClickHouse/issues/37895

-- end



DROP TABLE IF EXISTS gdp.ProtobufListProtodelim_kafka

Query id: 2cceac4a-f9ce-4a95-a831-2cea9c88c734

Ok.

0 rows in set. Elapsed: 0.001 sec.


CREATE TABLE IF NOT EXISTS gdp.ProtobufListProtodelim_kafka
(
          `Timestamp_Ns` DateTime64(9, 'UTC') CODEC(DoubleDelta, LZ4),
          `Hostname` LowCardinality(String) CODEC(LZ4),
          `Pop` LowCardinality(String) CODEC(LZ4),
          `Label` LowCardinality(String) CODEC(LZ4),
          `Tag` LowCardinality(String) CODEC(LZ4),
          `Poll_Counter` UInt64 CODEC(DoubleDelta, LZ4),
          `Record_Counter` UInt64 CODEC(DoubleDelta, LZ4),
          `Function` LowCardinality(String) CODEC(LZ4),
          `Variable` LowCardinality(String) CODEC(LZ4),
          `Type` LowCardinality(String) CODEC(LZ4),
          `Value` Float64
)
ENGINE = Kafka
SETTINGS kafka_broker_list = 'redpanda-0:9092', kafka_topic_list = 'ProtobufListProtodelim', kafka_schema =
'prometheus_protolist.proto:PromRecordCounter', kafka_max_rows_per_message = 10000, kafka_num_consumers = 1, kafka_thread_per_consumer = 0,
kafka_group_name = 'ProtobufListProtodelim', kafka_skip_broken_messages = 1, kafka_handle_error_mode = 'stream', kafka_format = 'ProtobufList'

Query id: 5e7f0dae-cad6-4d14-b94a-c169d6140883

Ok.

0 rows in set. Elapsed: 0.003 sec.

788427f2113d :) --
-- gdp.ProtobufListProtodelim_mv.sql
--
DROP VIEW IF EXISTS gdp.ProtobufListProtodelim_mv;

CREATE MATERIALIZED VIEW gdp.ProtobufListProtodelim_mv TO gdp.ProtobufListProtodelim
  AS SELECT
          *,
--        toDateTime64(Timestamp_Ns, 9, 'UTC') AS Timestamp_Ns,
--        Hostname,
--        Pop,
--        Label,
--        Tag,
--        Poll_Counter,
--        Record_Counter,
--        Function,
--        Variable,
--        Type,
--        Value,
  FROM gdp.ProtobufListProtodelim_kafka;
```

```
-- SHOW CREATE TABLE gdp.ProtobufListProtodelim_mv;

-- https://clickhouse.com/docs/sql-reference/statements/create/view#materialized-view

-- See also:
-- https://github.com/ClickHouse/ClickHouse/blob/master/tests/integration/test_storage_kafka/test_batch_fast.py#L2679
-- https://github.com/ClickHouse/ClickHouse/blob/master/tests/integration/test_storage_kafka/test_batch_slow.py

-- end



DROP VIEW IF EXISTS gdp.ProtobufListProtodelim_mv

Query id: 4276f466-8420-4d38-bcba-f97986b1e9ce

Ok.

0 rows in set. Elapsed: 0.001 sec.


CREATE MATERIALIZED VIEW gdp.ProtobufListProtodelim_mv TO gdp.ProtobufListProtodelim
AS SELECT *
FROM gdp.ProtobufListProtodelim_kafka

Query id: 8bbb212e-b13f-4763-9998-23cd8e111508

Ok.

0 rows in set. Elapsed: 0.004 sec.

788427f2113d :) show tables

SHOW TABLES

Query id: f9bb84e5-d59f-4507-89fc-f04fa97772fb

   ┌─name──────────────────────────┐
1. │ ProtobufListProtodelim        │
2. │ ProtobufListProtodelim_kafka  │
3. │ ProtobufListProtodelim_mv     │
   └───────────────────────────────┘

3 rows in set. Elapsed: 0.002 sec.
```

Creating the ProtobufListProtoDelim tables

Check kafka

```
788427f2113d :) SELECT * FROM system.kafka_consumers FORMAT Vertical;

SELECT *
FROM system.kafka_consumers
FORMAT Vertical

Query id: b3f05fc3-9e4f-4814-8a0b-dd800a65afe7

Row 1:
──────
database:                     gdp
table:                        ProtobufListProtodelim_kafka
consumer_id:                  ClickHouse-788427f2113d-gdp-ProtobufListProtodelim_kafka-a95aa8ac-34cc-4ac3-b14d-d1490bcd3bdd
assignments.topic:            ['ProtobufListProtodelim']
assignments.partition_id:   [0]
assignments.current_offset: [1519]
exceptions.time:              []
exceptions.text:              []
last_poll_time:               2025-04-06 23:54:31
num_messages_read:            1519
last_commit_time:             1970-01-01 00:00:00
num_commits:                  0
last_rebalance_time:          1970-01-01 00:00:00
num_rebalance_revocations:  0
num_rebalance_assignments:  1
is_currently_used:            1
last_used:                    1970-01-01 00:00:00.000000
rdkafka_stat:

1 row in set. Elapsed: 0.019 sec.
```

Check fafka, but it's not looking good. 😕
https://github.com/randomizedcoder/gdp/blob/main/build/containers/clickhouse/initdb.d/sql/gdp/ProtobufListProtodelim_kafka.sql#L32

Check the target table

```
788427f2113d :) SELECT * FROM gdp.ProtobufListProtodelim LIMIT 20;
```

```
SELECT *
FROM gdp.ProtobufListProtodelim
LIMIT 20

Query id: 845d7ee2-7c88-4a9d-9430-7640fa1e3d46

Ok.

0 rows in set. Elapsed: 0.009 sec.
```

Check target table
https://github.com/randomizedcoder/gdp/blob/main/build/containers/clickhouse/initdb.d/sql/gdp/ProtobufListProtodelim.sql#L27

You can also verify the Kafka counters here. We don't see erors, but it's not working. ??!

```
7392f49b6a57 :) SELECT
          event,
          value,
          description
FROM system.events
WHERE event LIKE '%Kafka%';

SELECT
          event,
          value,
          description
FROM system.events
WHERE event LIKE '%Kafka%'

Query id: 610725fb-b87d-442a-a19c-fc6374203279

   ┌─event─────────────────────┬─value─┬─description──────────────────────────────────────────────────────────────────────────────┐
1. │ KafkaRebalanceAssignments │     1 │ Number of partition assignments (the final stage of consumer group rebalance)        │
2. │ KafkaMessagesPolled       │  1525 │ Number of Kafka messages polled from librdkafka to ClickHouse                        │
3. │ KafkaMessagesRead         │     2 │ Number of Kafka messages already processed by ClickHouse                             │
4. │ KafkaRowsRead             │     5 │ Number of rows parsed from Kafka messages                                            │
5. │ KafkaBackgroundReads      │     1 │ Number of background reads populating materialized views from Kafka since server start │
   └───────────────────────────┴───────┴──────────────────────────────────────────────────────────────────────────────────────┘

5 rows in set. Elapsed: 0.002 sec.
```

Check Kafka system events
https://github.com/randomizedcoder/gdp/blob/main/build/containers/clickhouse/initdb.d/sql/select_system_events_like_kafka.sql

We also don't see errors in the error log

```
[das@t:~/Downloads/gdp]$ docker exec -ti gdp-clickhouse-1 tail -n 30 -f /var/log/clickhouse-server/clickhouse-server.err.log
2025.04.07 00:00:27.631662 [ 78 ] {} <Warning> AsynchronousMetrics: Hardware monitor 'thinkpad', sensor '8' exists but could not be read: errno: 6,
strerror: No such device or address.
2025.04.07 00:00:28.029035 [ 78 ] {} <Warning> Context: Delay accounting is not enabled, OSIOWaitMicroseconds will not be gathered. You can enable
it using `echo 1 > /proc/sys/kernel/task_delayacct` or by using sysctl.
2025.04.07 00:00:28.967224 [ 622 ] {} <Warning> StorageKafka (ProtobufListProtodelim_kafka): sasl.kerberos.kinit.cmd configuration parameter is
ignored.
2025.04.07 00:00:32.330922 [ 1 ] {} <Warning> AsynchronousMetrics: Hardware monitor 'thinkpad', sensor '8' exists but could not be read: errno: 6,
strerror: No such device or address.
2025.04.07 00:00:32.630508 [ 1 ] {} <Warning> Context: Delay accounting is not enabled, OSIOWaitMicroseconds will not be gathered. You can enable
it using `echo 1 > /proc/sys/kernel/task_delayacct` or by using sysctl.
2025.04.07 00:01:21.447752 [ 1392 ] {} <Warning> StorageKafka (ProtobufListProtodelim_kafka): sasl.kerberos.kinit.cmd configuration parameter is
ignored.
```

No errors. ?? So what's going on? = I have no idea

Works on some machines - OS and kernel versions

Actually it works on some machines

Kernel?	User	Works?	Comment

[das@t:~/Downloads/gdp/cmd]$ uname -a Linux t 6.12.21 #1-NixOS SMP PREEMPT_DYNAMIC Fri Mar 28 21:03:33 UTC 2025 x86_64 GNU/Linux	dave	No	ProtobufSingle works! ProtobufListProtodelim works if queue is small
[das@hp1:~/gdp]$ uname -a Linux hp1 6.13.7 #1-NixOS SMP PREEMPT_DYNAMIC Thu Mar 13 12:08:08 UTC 2025 x86_64 GNU/Linux	dave	No	ProtobufSingle works! ProtobufListProtodelim works if queue is small
[das@hp2:~/gdp]$ uname -a Linux hp2 6.12.10 #1-NixOS SMP PREEMPT_DYNAMIC Fri Jan 17 12:41:00 UTC 2025 x86_64 GNU/Linux		Yes!	ProtobufSingle works! ProtobufListProtodelim works if queue is small
# uname -a Linux devbox 6.11.0-21-generic #21~24.04.1-Ubuntu SMP PREEMPT_DYNAMIC Mon Feb 24 16:52:15 UTC 2 x86_64 x86_64 x86_64 GNU/Linux	anant	Yes!	ProtobufSingle works! ProtobufListProtodelim works if queue is small
Ubuntu 24.04.1 LTS 6.8.0-49-generic	Vivek	Yes!	ProtobufSingle works! ProtobufListProtodelim works if queue is small
Sequoia 15.4 on a 2018 intel Mac mini	Brent	Yes	ProtobufSingle works! ProtobufListProtodelim works if queue is small
MacOS Sequoia 15.3.2 on an M1 MacBook Pro	Brent		No :(I guess M1 will need some work

Long running

The machine hp2 was running for ~2 days and still had records. ~3k records in the ProtobufListProtodelim queue.

```
a23f089990ff :) select * from ProtobufListProtodelim limit 10;

SELECT *
FROM ProtobufListProtodelim
LIMIT 10

Query id: e868be44-63be-45d0-abdf-bde8754a2f3c

Connecting to database gdp at localhost:9000 as user default.
Connected to ClickHouse server version 25.3.1.


┌─────────────────Timestamp_Ns─┬─Hostname─────┬─Pop─┬─Label─┬─Tag─┬─Poll_Counter─┬─Record_Counter─┬─Function──────────────┬─Variable──────────┬─T
ype─────┬─Value─┐
 1. │ 2025-04-08 01:31:22.197552896 │ 7f5cdb221f73 │     │       │     │            0 │              0 │ Poller                │
pollingLoops        │ count   │         1 │
 2. │ 2025-04-08 01:31:22.197552896 │ 7f5cdb221f73 │     │       │     │            0 │              1 │ Poller                │
ticker              │ count   │         1 │
 3. │ 2025-04-08 01:31:22.197552896 │ 7f5cdb221f73 │     │       │     │            0 │              2 │ Run                   │ start
         │ counter │         1 │
 4. │ 2025-04-08 01:31:22.197552896 │ 7f5cdb221f73 │     │       │     │            0 │              3 │ fetchPrometheusMetrics │ start
         │ count   │         1 │
 5. │ 2025-04-08 01:31:22.197552896 │ 7f5cdb221f73 │     │       │     │            0 │              4 │ performPoll           │ start
         │ count   │         1 │
 6. │ 2025-04-08 01:32:22.207783424 │ 7f5cdb221f73 │     │       │     │            1 │              0 │ Poller                │
pollingLoops        │ count   │         2 │
 7. │ 2025-04-08 01:32:22.207783424 │ 7f5cdb221f73 │     │       │     │            1 │              1 │ Poller                │
ticker              │ count   │         2 │
```

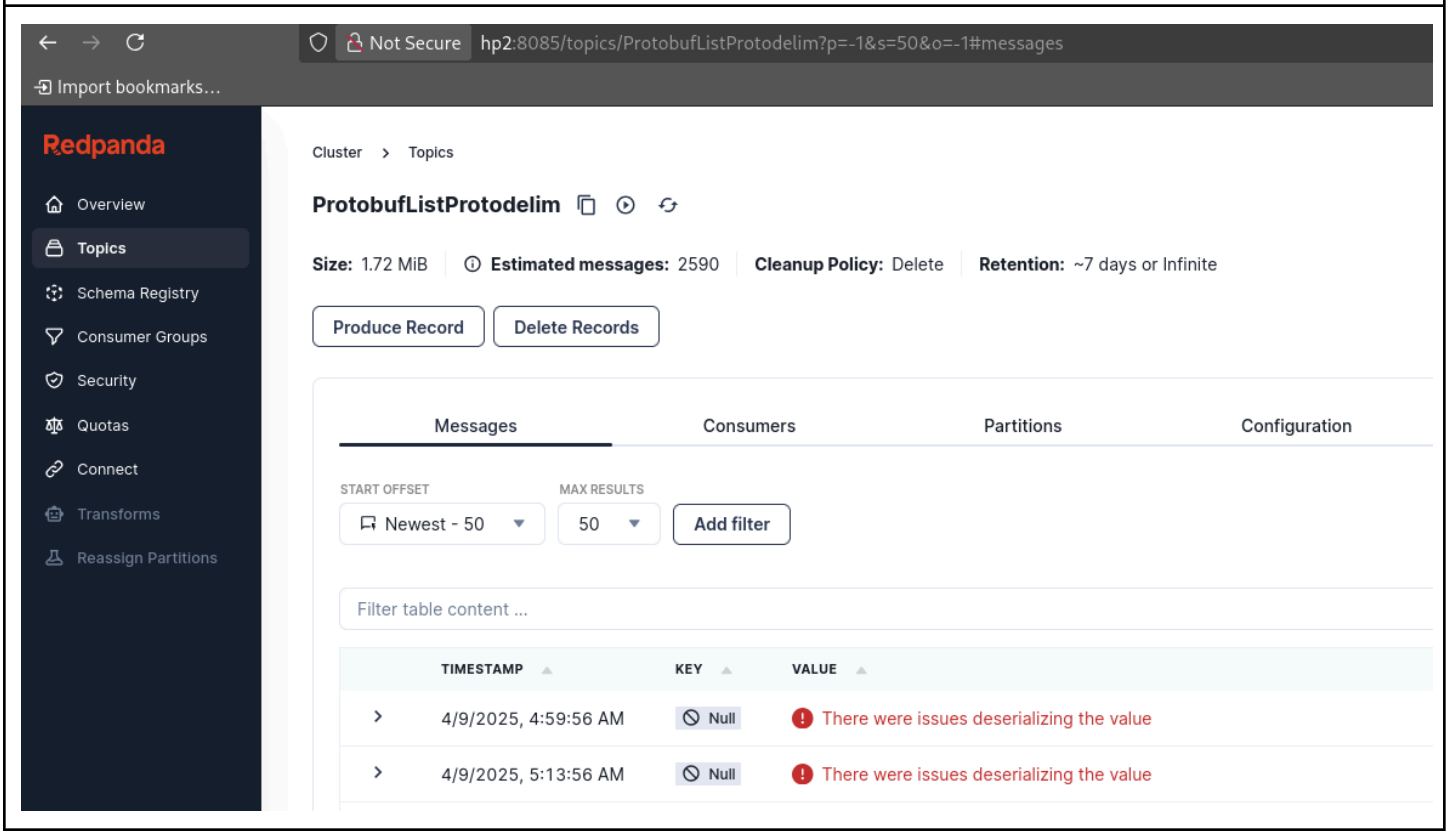
```
 8. | 2025-04-08 01:32:22.207783424 | 7f5cdb221f73 |        |         |         |         1 |                2 | Run                       | start
          | counter |           1 |
 9. | 2025-04-08 01:32:22.207783424 | 7f5cdb221f73 |        |         |         |         1 |                3 | destKafka                 |
ProtobufList        | count   |           1 |
10. | 2025-04-08 01:32:22.207783424 | 7f5cdb221f73 |        |         |         |         1 |                4 | destKafka                 |
ProtobufListKafka | count   |           1 |

└──────────────────────────────┴──────────────┴──────┴─────────┴──────┴─────────────┴──────────────────┴───────────────────────────┴──────────────────
──────┴─────────┘

10 rows in set. Elapsed: 0.005 sec. Processed 16.41 thousand rows, 641.40 KB (3.02 million rows/s., 118.12 MB/s.)
Peak memory usage: 575.69 KiB.
```

```
a23f089990ff :) select count(*) from ProtobufListProtodelim limit 10;

SELECT count(*)
FROM ProtobufListProtodelim
LIMIT 10

Query id: 55b642fe-5553-46d7-9ee2-4c69dcf9ea3a

   ┌─count()─┐
1. │   60037 │
   └─────────┘

1 row in set. Elapsed: 0.003 sec.
```

```
a23f089990ff :) SELECT
          count(DISTINCT Poll_Counter)
FROM gdp.ProtobufListProtodelim;

SELECT countDistinct(Poll_Counter)
FROM gdp.ProtobufListProtodelim

Query id: e14bb2a0-933c-4445-8ecb-6917319e3298

   ┌─countDistinct(Poll_Counter)─┐
1. │                        2147 │
   └─────────────────────────────┘

1 row in set. Elapsed: 0.007 sec. Processed 60.09 thousand rows, 480.74 KB (8.86 million rows/s., 70.92 MB/s.)
Peak memory usage: 93.80 KiB.
```

```
a23f089990ff :) SELECT * FROM system.kafka_consumers FORMAT Vertical;

SELECT *
FROM system.kafka_consumers
FORMAT Vertical

Query id: f3ad476b-174e-4ac0-a3ee-968fed0e80b2

Row 1:
──────
database:                   gdp
table:                      ProtobufListProtodelim_kafka
consumer_id:                ClickHouse-a23f089990ff-gdp-ProtobufListProtodelim_kafka-947407b9-dad1-49aa-8e70-385ce96fcac1
assignments.topic:          ['ProtobufListProtodelim']
assignments.partition_id:   [0]
assignments.current_offset: [2591]
exceptions.time:            []
exceptions.text:            []
last_poll_time:             2025-04-09 13:15:36
num_messages_read:          2144
last_commit_time:           2025-04-09 13:14:59
num_commits:                2144
last_rebalance_time:        1970-01-01 00:00:00
num_rebalance_revocations:  0
num_rebalance_assignments:  1
is_currently_used:          1
last_used:                  2025-04-09 13:15:32.466260
rdkafka_stat:               { "name": "ClickHouse-a23f089990ff-gdp-ProtobufListProtodelim_kafka#consumer-2", "client_id":
"ClickHouse-a23f089990ff-gdp-ProtobufListProtodelim_kafka", "type": "consumer", "ts":155732281996, "time":1744204536, "age":128644565120,
"replyq":0, "msg_cnt":0, "msg_size":0, "msg_max":0, "msg_size_max":0, "simple_cnt":0, "metadata_cache_cnt":1, "brokers":{ "redpanda-0:9092/0": {
"name":"redpanda-0:9092/0", "nodeid":0, "nodename":"redpanda-0:9092", "source":"configured", "state":"UP", "stateage":128644540132, "outbuf_cnt":0,
"outbuf_msg_cnt":0, "waitresp_cnt":1, "waitresp_msg_cnt":0, "tx":257864, "txbytes":41476050, "txerrs":0, "txretries":0, "txidle":109619,
"req_timeouts":0, "rx":257863, "rxbytes":25238763, "rxerrs":0, "rxcorriderrs":0, "rxpartial":0, "rxidle":109891, "zbuf_grow":0, "buf_grow":0,
"wakeups":642989, "connects":1, "disconnects":0, "int_latency": { "min":0, "max":0, "avg":0, "sum":0, "stddev": 0, "p50": 0, "p75": 0, "p90": 0,
"p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0, "cnt":0 }, "outbuf_latency": { "min":50, "max":132, "avg":91, "sum":547, "stddev":
0, "p50": 0, "p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0, "cnt":6 }, "rtt": { "min":500213, "max":500611,
"avg":500381, "sum":3002289, "stddev": 0, "p50": 0, "p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0, "cnt":6 },
"throttle": { "min":0, "max":0, "avg":0, "sum":0, "stddev": 0, "p50": 0, "p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0,
"hdrsize": 0, "cnt":0 }, "req": { "Fetch": 257218, "ListOffsets": 0, "Metadata": 429, "OffsetCommit": 0, "OffsetFetch": 0, "FindCoordinator": 216,
"JoinGroup": 0, "Heartbeat": 0, "LeaveGroup": 0, "SyncGroup": 0, "SaslHandshake": 0, "ApiVersion": 1, "SaslAuthenticate": 0, "DescribeCluster": 0,
"DescribeProducers": 0, "Unknown-62?": 0, "DescribeTransactions": 0, "ListTransactions": 0, "Unknown-69?": 0, "Unknown-70?": 0,
"GetTelemetrySubscriptions": 0, "PushTelemetry": 0, "Unknown-73?": 0 }, "toppars":{ "ProtobufListProtodelim-0": { "topic":"ProtobufListProtodelim",
"partition":0} } } , "GroupCoordinator": { "name":"GroupCoordinator", "nodeid":0, "nodename":"redpanda-0:9092", "source":"logical", "state":"UP",
"stateage":128644529916, "outbuf_cnt":1, "outbuf_msg_cnt":0, "waitresp_cnt":0, "waitresp_msg_cnt":0, "tx":47164, "txbytes":9141299, "txerrs":0,
"txretries":0, "txidle":3001120, "req_timeouts":0, "rx":47164, "rxbytes":834959, "rxerrs":0, "rxcorriderrs":0, "rxpartial":0, "rxidle":3000946,
"zbuf_grow":0, "buf_grow":0, "wakeups":227423, "connects":1, "disconnects":0, "int_latency": { "min":0, "max":0, "avg":0, "sum":0, "stddev": 0,
"p50": 0, "p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0, "cnt":0 }, "outbuf_latency": { "min":73, "max":73,
"avg":73, "sum":73, "stddev": 0, "p50": 0, "p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0, "cnt":1 }, "rtt": {
"min":174, "max":174, "avg":174, "sum":174, "stddev": 0, "p50": 0, "p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize":
0, "cnt":1 }, "throttle": { "min":0, "max":0, "avg":0, "sum":0, "stddev": 0, "p50": 0, "p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0,
"outofrange": 0, "hdrsize": 0, "cnt":0 }, "req": { "Fetch": 0, "ListOffsets": 0, "Metadata": 1, "OffsetCommit": 2144, "OffsetFetch": 2145,
"FindCoordinator": 0, "JoinGroup": 2, "Heartbeat": 42870, "LeaveGroup": 0, "SyncGroup": 1, "SaslHandshake": 0, "ApiVersion": 1, "SaslAuthenticate":
0, "DescribeCluster": 0, "DescribeProducers": 0, "Unknown-62?": 0, "DescribeTransactions": 0, "ListTransactions": 0, "Unknown-69?": 0,
"Unknown-70?": 0, "GetTelemetrySubscriptions": 0, "PushTelemetry": 0, "Unknown-73?": 0 }, "toppars":{ } } }, "topics":{ "ProtobufListProtodelim": {
"topic":"ProtobufListProtodelim", "age":128644504, "metadata_age":244175, "batchsize": { "min":0, "max":0, "avg":0, "sum":0, "stddev": 0, "p50": 0,
"p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0, "cnt":0 }, "batchcnt": { "min":0, "max":0, "avg":0, "sum":0,
"stddev": 0, "p50": 0, "p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0, "cnt":0 }, "partitions":{ "0": {
"partition":0, "broker":0, "leader":0, "desired":true, "unknown":false, "msgq_cnt":0, "msgq_bytes":0, "xmit_msgq_cnt":0, "xmit_msgq_bytes":0,
"fetchq_cnt":0, "fetchq_size":0, "fetch_state":"active", "query_offset":-1001, "next_offset":2591, "app_offset":2591, "stored_offset":2591,
```

```
"stored_leader_epoch":-1, "commited_offset":2591, "committed_offset":2591, "committed_leader_epoch":-1, "eof_offset":2591, "lo_offset":0,
"hi_offset":2591, "ls_offset":2591, "consumer_lag":0, "consumer_lag_stored":0, "leader_epoch":-1, "txmsgs":0, "txbytes":0, "rxmsgs":2144,
"rxbytes":4914976, "msgs": 2144, "rx_ver_drops": 0, "msgs_inflight": 0, "next_ack_seq": 0, "next_err_seq": 0, "acked_msgid": 0} , "-1": {
"partition":-1, "broker":-1, "leader":-1, "desired":false, "unknown":false, "msgq_cnt":0, "msgq_bytes":0, "xmit_msgq_cnt":0, "xmit_msgq_bytes":0,
"fetchq_cnt":0, "fetchq_size":0, "fetch_state":"none", "query_offset":-1001, "next_offset":0, "app_offset":-1001, "stored_offset":-1001,
"stored_leader_epoch":-1, "commited_offset":-1001, "committed_offset":-1001, "committed_leader_epoch":-1, "eof_offset":-1001, "lo_offset":-1001,
"hi_offset":-1001, "ls_offset":-1001, "consumer_lag":-1, "consumer_lag_stored":-1, "leader_epoch":-1, "txmsgs":0, "txbytes":0, "rxmsgs":0,
"rxbytes":0, "msgs": 0, "rx_ver_drops": 0, "msgs_inflight": 0, "next_ack_seq": 0, "next_err_seq": 0, "acked_msgid": 0} } } } , "cgrp": { "state":
"up", "stateage": 128644529, "join_state": "steady", "rebalance_age": 128644504, "rebalance_cnt": 1, "rebalance_reason": "Metadata for subscribed
topic(s) has changed", "assignment_size": 1 }, "tx":305028, "tx_bytes":50617349, "rx":305027, "rx_bytes":26073722, "txmsgs":0, "txmsg_bytes":0,
"rxmsgs":2144, "rxmsg_bytes":4914976}
```

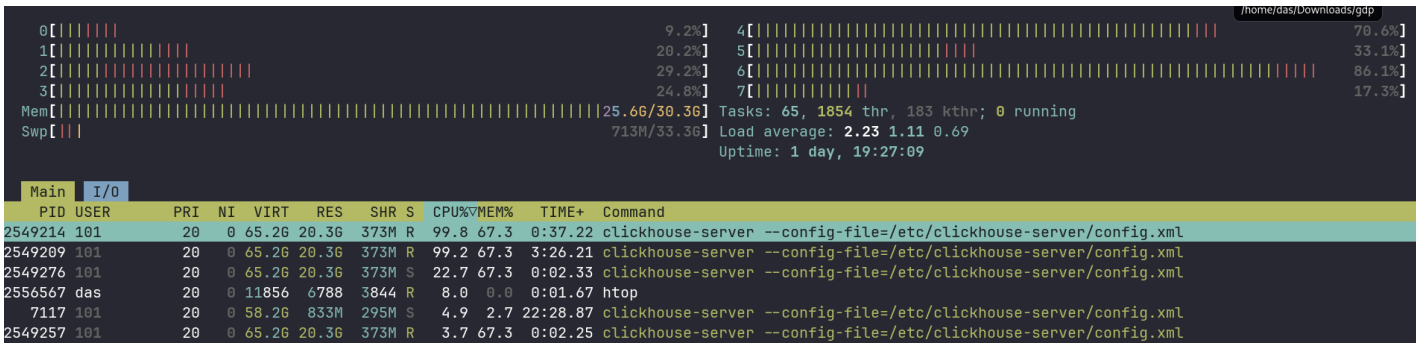



Let’s restart Clickhouse, so we will recreate the database, and Clickhouse will need to reingest all the data.

```
[das@hp2:~/gdp]$ make build_clickhouse_and_deploy
================================
Make builddocker_clickhouse randomizedcoder/gdp_clickhouse:1.0.0
docker build \
                --progress=plain \
                --no-cache \
                --build-arg GDPPATH=/home/das/gdp \
                --build-arg VERSION=1.0.0 \
                --file build/containers/clickhouse/Containerfile \
                --tag randomizedcoder/gdp_clickhouse:1.0.0 --tag randomizedcoder/gdp_clickhouse:latest \
                .

#0 building with "default" instance using docker driver

#1 [internal] load build definition from Containerfile
#1 transferring dockerfile: 2.31kB done
#1 DONE 0.0s

#2 [internal] load metadata for docker.io/clickhouse/clickhouse-server:25.3.1
#2 DONE 0.7s

#3 [internal] load .dockerignore
#3 transferring context: 77B done
#3 DONE 0.0s

#4 [1/7] FROM docker.io/clickhouse/clickhouse-server:25.3.1@sha256:ace26695dde5d882625ef43132e8caeff758012925852e9fe76bbb6d2bf3a61b
#4 CACHED

#5 [internal] load build context
#5 transferring context: 325B done
#5 DONE 0.0s

#6 [2/7] RUN echo GDPPATH:/home/das/gdp VERSION:1.0.0
#6 0.197 GDPPATH:/home/das/gdp VERSION:1.0.0
#6 DONE 0.3s

#7 [3/7] COPY --chmod=644       ./build/containers/clickhouse/users.xml         /etc/clickhouse-server/users.xml
#7 DONE 0.0s

#8 [4/7] COPY --chmod=644       ./build/containers/clickhouse/config.xml        /etc/clickhouse-server/config.xml
#8 DONE 0.1s

#9 [5/7] COPY --chmod=644       ./build/containers/clickhouse/docker_related_config.xml         /etc/clickhouse-server/config.d/docker_related_config.xml
#9 DONE 0.1s

#10 [6/7] COPY --chmod=644      ./VERSION       /VERSION
#10 DONE 0.0s

#11 [7/7] RUN cat /VERSION
#11 0.190 1.0.0
#11 DONE 0.2s

#12 exporting to image
#12 exporting layers 0.1s done
#12 writing image sha256:3dd9442aaba0332c3954d6b6f6920d79528dc87bb33af3152257779f4fafef38 done
#12 naming to docker.io/randomizedcoder/gdp_clickhouse:1.0.0 done
#12 naming to docker.io/randomizedcoder/gdp_clickhouse:latest done
#12 DONE 0.1s
./check_protos.bash
running check_protos.bash
```

```
compare_files ./proto/prometheus/v1/prometheus.proto ./build/containers/clickhouse/format_schemas/prometheus.proto
diff_result is zero: ./proto/prometheus/v1/prometheus.proto:./build/containers/clickhouse/format_schemas/prometheus.proto
compare_files ./proto/prometheus/v1/prometheus_protolist.proto ./build/containers/clickhouse/format_schemas/prometheus_protolist.proto
diff_result is zero: ./proto/prometheus/v1/prometheus_protolist.proto:./build/containers/clickhouse/format_schemas/prometheus_protolist.proto
compare_files ./proto/clickhouse_protolist/v1/clickhouse_protolist.proto ./build/containers/clickhouse/format_schemas/clickhouse_protolist.proto
diff_result is zero: ./proto/clickhouse_protolist/v1/clickhouse_protolist.proto:./build/containers/clickhouse/format_schemas/clickhouse_protolist.proto
================================
Make deploy
echo GDPPATH=/home/das/gdp
GDPPATH=/home/das/gdp
GDPPATH=/home/das/gdp \
docker compose \
            --file build/containers/docker-compose.yml \
            up -d --remove-orphans
WARN[0000] mount of type `volume` should not define `bind` option
[+] Running 4/4
 ✔ Container redpanda-0                      Running
            0.0s
 ✔ Container gdp-redpanda-console-1  Running
            0.0s
 ✔ Container gdp-gdp-1                       Running
            0.0s
 ✔ Container gdp-clickhouse-1                Started
            5.0s
Commands to try next:
-------
docker logs --follow gdp-gdp-1
docker logs --follow gdp-clickhouse-1
docker exec -ti gdp-clickhouse-1 clickhouse-client
-------
assuming the gdp distroless:debug is available
docker exec -ti gdp-gdp-1 sh
docker exec -ti gdp-clickhouse-1 bash
clickhouse: sudo apt install iputils-ping
------
docker exec -ti gdp-clickhouse-1 tail -n 30 -f /var/log/clickhouse-server/clickhouse-server.err.log
docker exec -ti gdp-clickhouse-1 tail -n 30 -f /var/log/clickhouse-server/clickhouse-server.log
docker exec -ti gdp-clickhouse-1 clickhouse-client
docker exec -ti gdp-clickhouse-1 clickhouse-client --query "SELECT count(*) FROM gdp.gdp;"
docker exec -ti gdp-clickhouse-1 clickhouse-client --query "SELECT * FROM system.kafka_consumers FORMAT Vertical;"
docker exec -ti gdp-clickhouse-1 clickhouse-client --query "SELECT * FROM system.stack_trace ORDER BY 'thread_id' DESC LIMIT 10;"
------
Browse: http://localhost:8085/topics/topic?p=-1&s=50&o=-2#messages
```

CPU goes very high



Error log. Lots of this repeated

```
[das@hp2:~/gdp]$ docker exec -ti gdp-clickhouse-1 tail -n 30 -f
/var/log/clickhouse-server/clickhouse-server.err.log
2025.04.09 13:37:48.862409 [ 642 ] {} <Warning> StorageKafka (ProtobufSingle_kafka): Can't get assignment.
Will keep trying.
2025.04.09 13:37:49.416070 [ 643 ] {} <Warning> StorageKafka (ProtobufSingle_kafka): Can't get assignment.
Will keep trying.
2025.04.09 13:37:49.969697 [ 644 ] {} <Warning> StorageKafka (ProtobufSingle_kafka): Can't get assignment.
Will keep trying.
2025.04.09 13:37:50.522756 [ 645 ] {} <Warning> StorageKafka (ProtobufSingle_kafka): Can't get assignment.
Will keep trying.
2025.04.09 13:37:51.075088 [ 646 ] {} <Warning> StorageKafka (ProtobufSingle_kafka): Can't get assignment.
Will keep trying.
2025.04.09 13:37:51.628944 [ 647 ] {} <Warning> StorageKafka (ProtobufSingle_kafka): Can't get assignment.
Will keep trying.
2025.04.09 13:37:52.181487 [ 648 ] {} <Warning> StorageKafka (ProtobufSingle_kafka): Can't get assignment.
Will keep trying.
2025.04.09 13:37:52.735341 [ 649 ] {} <Warning> StorageKafka (ProtobufSingle_kafka): Can't get assignment.
Will keep trying.
2025.04.09 13:37:53.289384 [ 650 ] {} <Warning> StorageKafka (ProtobufSingle_kafka): Can't get assignment.
Will keep trying.
2025.04.09 13:37:53.843614 [ 651 ] {} <Warning> StorageKafka (ProtobufSingle_kafka): Can't get assignment.
Will keep trying.
2025.04.09 13:37:54.397476 [ 652 ] {} <Warning> StorageKafka (ProtobufSingle_kafka): Can't get assignment.
Will keep trying.
2025.04.09 13:37:54.951413 [ 653 ] {} <Warning> StorageKafka (ProtobufSingle_kafka): Can't get assignment.
Will keep trying.
```

```
d2775600408f :) SELECT * FROM system.kafka_consumers FORMAT Vertical;
```

```
SELECT *
FROM system.kafka_consumers
FORMAT Vertical

Query id: 7e2452de-57c5-4a53-b396-5fabf10a128a

Row 1:
──────
database:                     gdp
table:                        ProtobufListProtodelim_kafka
consumer_id:                  ClickHouse-d2775600408f-gdp-ProtobufListProtodelim_kafka-bbe350c7-93a8-4496-97ef-314651a7a777
assignments.topic:            ['ProtobufListProtodelim']
assignments.partition_id:   [0]
assignments.current_offset: [10]
exceptions.time:              ['2025-04-09 14:42:55','2025-04-09 14:42:55','2025-04-09 14:42:55','2025-04-09 14:42:55','2025-04-09
14:42:55','2025-04-09 14:42:55','2025-04-09 14:42:56']
exceptions.text:              ['Local: Group partition assignment lost0. DB::KafkaConsumer::setExceptionInfo(String const&, bool) @
0x0000000011ce4090\n1. DB::KafkaConsumer::commit() @ 0x0000000011ce1fec\n2. DB::StorageKafka::threadFunc(unsigned long) @ 0x0000000011cfad48\n3.
DB::BackgroundSchedulePool::threadFunction() @ 0x00000000125376c3\n4. void std::__function::__policy_invoker<void
()>::__call_impl[abi:ne190107]<std::__function::__default_alloc_func<ThreadFromGlobalPoolImpl<false,
true>::ThreadFromGlobalPoolImpl<DB::BackgroundSchedulePool::BackgroundSchedulePool(unsigned long, StrongTypedef<unsigned long,
CurrentMetrics::MetricTag>, StrongTypedef<unsigned long, CurrentMetrics::MetricTag>, char
const*)::$_1>(DB::BackgroundSchedulePool::BackgroundSchedulePool(unsigned long, StrongTypedef<unsigned long, CurrentMetrics::MetricTag>,
StrongTypedef<unsigned long, CurrentMetrics::MetricTag>, char const*)::$_1&&)::\'lambda\'(), void ()>>(std::__function::__policy_storage const*) @
0x0000000012538ca7\n5. ThreadPoolImpl<std::thread>::ThreadFromThreadPool::worker() @ 0x000000000f5c19ef\n6. void*
std::__thread_proxy[abi:ne190107]<std::tuple<std::unique_ptr<std::__thread_struct, std::default_delete<std::__thread_struct>>, void
(ThreadPoolImpl<std::thread>::ThreadFromThreadPool::*)(), ThreadPoolImpl<std::thread>::ThreadFromThreadPool*>>(void*) @ 0x000000000f5c8cba\n7. ? @
0x00007fd4a66a5ac3\n8. ? @ 0x00007fd4a6736a04\n','Local: Group partition assignment lost0. DB::KafkaConsumer::setExceptionInfo(String const&, bool)
@ 0x0000000011ce4090\n1. DB::KafkaConsumer::commit() @ 0x0000000011ce1fec\n2. DB::StorageKafka::threadFunc(unsigned long) @ 0x0000000011cfad48\n3.
DB::BackgroundSchedulePool::threadFunction() @ 0x00000000125376c3\n4. void std::__function::__policy_invoker<void
()>::__call_impl[abi:ne190107]<std::__function::__default_alloc_func<ThreadFromGlobalPoolImpl<false,
true>::ThreadFromGlobalPoolImpl<DB::BackgroundSchedulePool::BackgroundSchedulePool(unsigned long, StrongTypedef<unsigned long,
CurrentMetrics::MetricTag>, StrongTypedef<unsigned long, CurrentMetrics::MetricTag>, char
const*)::$_1>(DB::BackgroundSchedulePool::BackgroundSchedulePool(unsigned long, StrongTypedef<unsigned long, CurrentMetrics::MetricTag>,
StrongTypedef<unsigned long, CurrentMetrics::MetricTag>, char const*)::$_1&&)::\'lambda\'(), void ()>>(std::__function::__policy_storage const*) @
0x0000000012538ca7\n5. ThreadPoolImpl<std::thread>::ThreadFromThreadPool::worker() @ 0x000000000f5c19ef\n6. void*
std::__thread_proxy[abi:ne190107]<std::tuple<std::unique_ptr<std::__thread_struct, std::default_delete<std::__thread_struct>>, void
(ThreadPoolImpl<std::thread>::ThreadFromThreadPool::*)(), ThreadPoolImpl<std::thread>::ThreadFromThreadPool*>>(void*) @ 0x000000000f5c8cba\n7. ? @
0x00007fd4a66a5ac3\n8. ? @ 0x00007fd4a6736a04\n','Local: Group partition assignment lost0. DB::KafkaConsumer::setExceptionInfo(String const&, bool)
@ 0x0000000011ce4090\n1. DB::KafkaConsumer::commit() @ 0x0000000011ce1fec\n2. DB::StorageKafka::threadFunc(unsigned long) @ 0x0000000011cfad48\n3.
DB::BackgroundSchedulePool::threadFunction() @ 0x00000000125376c3\n4. void std::__function::__policy_invoker<void
()>::__call_impl[abi:ne190107]<std::__function::__default_alloc_func<ThreadFromGlobalPoolImpl<false,
true>::ThreadFromGlobalPoolImpl<DB::BackgroundSchedulePool::BackgroundSchedulePool(unsigned long, StrongTypedef<unsigned long,
CurrentMetrics::MetricTag>, StrongTypedef<unsigned long, CurrentMetrics::MetricTag>, char
const*)::$_1>(DB::BackgroundSchedulePool::BackgroundSchedulePool(unsigned long, StrongTypedef<unsigned long, CurrentMetrics::MetricTag>,
StrongTypedef<unsigned long, CurrentMetrics::MetricTag>, char const*)::$_1&&)::\'lambda\'(), void ()>>(std::__function::__policy_storage const*) @
0x0000000012538ca7\n5. ThreadPoolImpl<std::thread>::ThreadFromThreadPool::worker() @ 0x000000000f5c19ef\n6. void*
std::__thread_proxy[abi:ne190107]<std::tuple<std::unique_ptr<std::__thread_struct, std::default_delete<std::__thread_struct>>, void
(ThreadPoolImpl<std::thread>::ThreadFromThreadPool::*)(), ThreadPoolImpl<std::thread>::ThreadFromThreadPool*>>(void*) @ 0x000000000f5c8cba\n7. ? @
0x00007fd4a66a5ac3\n8. ? @ 0x00007fd4a6736a04\n','Local: Group partition assignment lost0. DB::KafkaConsumer::setExceptionInfo(String const&, bool)
@ 0x0000000011ce4090\n1. DB::KafkaConsumer::commit() @ 0x0000000011ce1fec\n2. DB::StorageKafka::threadFunc(unsigned long) @ 0x0000000011cfad48\n3.
DB::BackgroundSchedulePool::threadFunction() @ 0x00000000125376c3\n4. void std::__function::__policy_invoker<void
()>::__call_impl[abi:ne190107]<std::__function::__default_alloc_func<ThreadFromGlobalPoolImpl<false,
true>::ThreadFromGlobalPoolImpl<DB::BackgroundSchedulePool::BackgroundSchedulePool(unsigned long, StrongTypedef<unsigned long,
CurrentMetrics::MetricTag>, StrongTypedef<unsigned long, CurrentMetrics::MetricTag>, char
const*)::$_1>(DB::BackgroundSchedulePool::BackgroundSchedulePool(unsigned long, StrongTypedef<unsigned long, CurrentMetrics::MetricTag>,
StrongTypedef<unsigned long, CurrentMetrics::MetricTag>, char const*)::$_1&&)::\'lambda\'(), void ()>>(std::__function::__policy_storage const*) @
0x0000000012538ca7\n5. ThreadPoolImpl<std::thread>::ThreadFromThreadPool::worker() @ 0x000000000f5c19ef\n6. void*
std::__thread_proxy[abi:ne190107]<std::tuple<std::unique_ptr<std::__thread_struct, std::default_delete<std::__thread_struct>>, void
(ThreadPoolImpl<std::thread>::ThreadFromThreadPool::*)(), ThreadPoolImpl<std::thread>::ThreadFromThreadPool*>>(void*) @ 0x000000000f5c8cba\n7. ? @
0x00007fd4a66a5ac3\n8. ? @ 0x00007fd4a6736a04\n','Local: Group partition assignment lost0. DB::KafkaConsumer::setExceptionInfo(String const&, bool)
@ 0x0000000011ce4090\n1. DB::KafkaConsumer::commit() @ 0x0000000011ce1fec\n2. DB::StorageKafka::threadFunc(unsigned long) @ 0x0000000011cfad48\n3.
DB::BackgroundSchedulePool::threadFunction() @ 0x00000000125376c3\n4. void std::__function::__policy_invoker<void
()>::__call_impl[abi:ne190107]<std::__function::__default_alloc_func<ThreadFromGlobalPoolImpl<false,
true>::ThreadFromGlobalPoolImpl<DB::BackgroundSchedulePool::BackgroundSchedulePool(unsigned long, StrongTypedef<unsigned long,
CurrentMetrics::MetricTag>, StrongTypedef<unsigned long, CurrentMetrics::MetricTag>, char
const*)::$_1>(DB::BackgroundSchedulePool::BackgroundSchedulePool(unsigned long, StrongTypedef<unsigned long, CurrentMetrics::MetricTag>,
StrongTypedef<unsigned long, CurrentMetrics::MetricTag>, char const*)::$_1&&)::\'lambda\'(), void ()>>(std::__function::__policy_storage const*) @
0x0000000012538ca7\n5. ThreadPoolImpl<std::thread>::ThreadFromThreadPool::worker() @ 0x000000000f5c19ef\n6. void*
std::__thread_proxy[abi:ne190107]<std::tuple<std::unique_ptr<std::__thread_struct, std::default_delete<std::__thread_struct>>, void
(ThreadPoolImpl<std::thread>::ThreadFromThreadPool::*)(), ThreadPoolImpl<std::thread>::ThreadFromThreadPool*>>(void*) @ 0x000000000f5c8cba\n7. ? @
0x00007fd4a66a5ac3\n8. ? @ 0x00007fd4a6736a04\n','Code: 518. DB::Exception: All commit attempts failed. Last block was already written to target
table(s), but was not committed to Kafka. (CANNOT_COMMIT_OFFSET), Stack trace (when copying this message, always include the lines below):\n\n0.
DB::Exception::Exception(DB::Exception::MessageMasked&&, int, bool) @ 0x000000000f48e57b\n1. DB::Exception::Exception(PreformattedMessage&&, int) @
0x0000000009f65a2c\n2. DB::Exception::Exception<>(int, FormatStringHelperImpl<>) @ 0x0000000009f721eb\n3. DB::KafkaConsumer::commit() @
0x0000000011ce281b\n4. DB::StorageKafka::threadFunc(unsigned long) @ 0x0000000011cfad48\n5. DB::BackgroundSchedulePool::threadFunction() @
0x00000000125376c3\n6. void std::__function::__policy_invoker<void
()>::__call_impl[abi:ne190107]<std::__function::__default_alloc_func<ThreadFromGlobalPoolImpl<false,
true>::ThreadFromGlobalPoolImpl<DB::BackgroundSchedulePool::BackgroundSchedulePool(unsigned long, StrongTypedef<unsigned long,
CurrentMetrics::MetricTag>, StrongTypedef<unsigned long, CurrentMetrics::MetricTag>, char
const*)::$_1>(DB::BackgroundSchedulePool::BackgroundSchedulePool(unsigned long, StrongTypedef<unsigned long, CurrentMetrics::MetricTag>,
StrongTypedef<unsigned long, CurrentMetrics::MetricTag>, char const*)::$_1&&)::\'lambda\'(), void ()>>(std::__function::__policy_storage const*) @
0x0000000012538ca7\n7. ThreadPoolImpl<std::thread>::ThreadFromThreadPool::worker() @ 0x000000000f5c19ef\n8. void*
std::__thread_proxy[abi:ne190107]<std::tuple<std::unique_ptr<std::__thread_struct, std::default_delete<std::__thread_struct>>, void
(ThreadPoolImpl<std::thread>::ThreadFromThreadPool::*)(), ThreadPoolImpl<std::thread>::ThreadFromThreadPool*>>(void*) @ 0x000000000f5c8cba\n9. ? @
0x00007fd4a66a5ac3\n10. ? @ 0x00007fd4a6736a04\n (version 25.3.1.2703 (official build))','Local: Maximum application poll interval
(max.poll.interval.ms) exceeded0. DB::KafkaConsumer::setExceptionInfo(String const&, bool) @ 0x0000000011ce4090\n1. void
std::__function::__policy_invoker<void (cppkafka::Error
const&)>::__call_impl[abi:ne190107]<std::__function::__default_alloc_func<DB::KafkaConsumer::doPoll()::$_0, void (cppkafka::Error
const&)>>(std::__function::__policy_storage const*, cppkafka::Error const&) @ 0x0000000011cf05be\n2.
_ZZN2DB17StorageKafkaUtils18eraseMessageErrorsERNSt3__16vectorIN8cppkafka7MessageENS1_9allocatorIS4_EEEERKNS1_10shared_ptrIN4Poco6LoggerEEENS1_8fun
ctionIFvRKNS3_5ErrorEEEEENK3$_0clIS4_EEDaRT_ @ 0x0000000011d5e13f\n3. DB::StorageKafkaUtils::eraseMessageErrors(std::vector<cppkafka::Message,
std::allocator<cppkafka::Message>>&, std::shared_ptr<Poco::Logger> const&, std::function<void (cppkafka::Error const&)>) @ 0x0000000011d5b27d\n4.
DB::KafkaConsumer::doPoll() @ 0x0000000011ce7739\n5. DB::KafkaConsumer::subscribe() @ 0x0000000011ce686f\n6. DB::KafkaSource::generateImpl() @
0x0000000011d539e5\n7. DB::KafkaSource::generate() @ 0x0000000011d5746f\n8. DB::ISource::tryGenerate() @ 0x0000000014cca29e\n9. DB::ISource::work()
@ 0x0000000014cc9ea7\n10. DB::ExecutionThreadContext::executeTask() @ 0x0000000014ce5a96\n11. DB::PipelineExecutor::executeStepImpl(unsigned long,
std::atomic<bool>*) @ 0x0000000014cd9584\n12. DB::PipelineExecutor::execute(unsigned long, bool) @ 0x0000000014cd875e\n13.
DB::CompletedPipelineExecutor::execute() @ 0x0000000014cd6faf\n14. DB::StorageKafka::threadFunc(unsigned long) @ 0x0000000011cfacd5\n15.
DB::BackgroundSchedulePool::threadFunction() @ 0x00000000125376c3\n16. void std::__function::__policy_invoker<void
()>::__call_impl[abi:ne190107]<std::__function::__default_alloc_func<ThreadFromGlobalPoolImpl<false,
true>::ThreadFromGlobalPoolImpl<DB::BackgroundSchedulePool::BackgroundSchedulePool(unsigned long, StrongTypedef<unsigned long,
CurrentMetrics::MetricTag>, StrongTypedef<unsigned long, CurrentMetrics::MetricTag>, char
const*)::$_1>(DB::BackgroundSchedulePool::BackgroundSchedulePool(unsigned long, StrongTypedef<unsigned long, CurrentMetrics::MetricTag>,
```

```
StrongTypedef<unsigned long, CurrentMetrics::MetricTag>, char const*)::$_1&&)::\'lambda\'(), void ()>>(std::__function::__policy_storage const*) @
0x0000000012538ca7\n17. ThreadPoolImpl<std::thread>::ThreadFromThreadPool::worker() @ 0x000000000f5c19ef\n18. void*
std::__thread_proxy[abi:ne190107]<std::tuple<std::unique_ptr<std::__thread_struct, std::default_delete<std::__thread_struct>>, void
(ThreadPoolImpl<std::thread>::ThreadFromThreadPool::*)(), ThreadPoolImpl<std::thread>::ThreadFromThreadPool*>>(void*) @ 0x000000000f5c8cba\n19. ? @
0x00007fd4a66a5ac3\n20. ? @ 0x00007fd4a6736a04\n']
last_poll_time:               2025-04-09 14:42:56
num_messages_read:            29
last_commit_time:             1970-01-01 00:00:00
num_commits:                  0
last_rebalance_time:          2025-04-09 14:42:56
num_rebalance_revocations:  1
num_rebalance_assignments:  2
is_currently_used:            1
last_used:                    2025-04-09 14:42:56.146951
rdkafka_stat:                 { "name": "ClickHouse-d2775600408f-gdp-ProtobufListProtodelim_kafka#consumer-1", "client_id":
"ClickHouse-d2775600408f-gdp-ProtobufListProtodelim_kafka", "type": "consumer", "ts":160970601940, "time":1744209774, "age":1968326231,
"replyq":4816, "msg_cnt":0, "msg_size":0, "msg_max":0, "msg_size_max":0, "simple_cnt":0, "metadata_cache_cnt":1, "brokers":{ "redpanda-0:9092/0": {
"name":"redpanda-0:9092/0", "nodeid":0, "nodename":"redpanda-0:9092", "source":"configured", "state":"UP", "stateage":1968320175, "outbuf_cnt":0,
"outbuf_msg_cnt":0, "waitresp_cnt":0, "waitresp_msg_cnt":0, "tx":2684, "txbytes":431278, "txerrs":0, "txretries":0, "txidle":168209725,
"req_timeouts":0, "rx":2684, "rxbytes":3150721, "rxerrs":0, "rxcorriderrs":0, "rxpartial":0, "rxidle":168209564, "zbuf_grow":0, "buf_grow":0,
"wakeups":7349, "connects":1, "disconnects":0, "int_latency": { "min":0, "max":0, "avg":0, "sum":0, "stddev": 0, "p50": 0, "p75": 0, "p90": 0,
"p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0, "cnt":0 }, "outbuf_latency": { "min":0, "max":0, "avg":0, "sum":0, "stddev": 0,
"p50": 0, "p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0, "cnt":0 }, "rtt": { "min":0, "max":0, "avg":0,
"sum":0, "stddev": 0, "p50": 0, "p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0, "cnt":0 }, "throttle": {
"min":0, "max":0, "avg":0, "sum":0, "stddev": 0, "p50": 0, "p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0,
"cnt":0 }, "req": { "Fetch": 2670, "ListOffsets": 1, "Metadata": 7, "OffsetCommit": 0, "OffsetFetch": 0, "FindCoordinator": 5, "JoinGroup": 0,
"Heartbeat": 0, "LeaveGroup": 0, "SyncGroup": 0, "SaslHandshake": 0, "ApiVersion": 1, "SaslAuthenticate": 0, "DescribeCluster": 0,
"DescribeProducers": 0, "Unknown-62?": 0, "DescribeTransactions": 0, "ListTransactions": 0, "Unknown-69?": 0, "Unknown-70?": 0,
"GetTelemetrySubscriptions": 0, "PushTelemetry": 0, "Unknown-73?": 0 }, "toppars":{ "ProtobufListProtodelim-0": { "topic":"ProtobufListProtodelim",
"partition":0} } } , "GroupCoordinator": { "name":"GroupCoordinator", "nodeid":0, "nodename":"redpanda-0:9092", "source":"logical", "state":"UP",
"stateage":1968317621, "outbuf_cnt":0, "outbuf_msg_cnt":0, "waitresp_cnt":0, "waitresp_msg_cnt":0, "tx":308, "txbytes":60117, "txerrs":0,
"txretries":0, "txidle":1067882645, "req_timeouts":0, "rx":308, "rxbytes":5311, "rxerrs":0, "rxcorriderrs":0, "rxpartial":0, "rxidle":1067882425,
"zbuf_grow":0, "buf_grow":0, "wakeups":2595, "connects":1, "disconnects":0, "int_latency": { "min":0, "max":0, "avg":0, "sum":0, "stddev": 0,
"p50": 0, "p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0, "cnt":0 }, "outbuf_latency": { "min":0, "max":0,
"avg":0, "sum":0, "stddev": 0, "p50": 0, "p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0, "cnt":0 }, "rtt": {
"min":0, "max":0, "avg":0, "sum":0, "stddev": 0, "p50": 0, "p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0,
"cnt":0 }, "throttle": { "min":0, "max":0, "avg":0, "sum":0, "stddev": 0, "p50": 0, "p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0,
"outofrange": 0, "hdrsize": 0, "cnt":0 }, "req": { "Fetch": 0, "ListOffsets": 0, "Metadata": 1, "OffsetCommit": 0, "OffsetFetch": 1,
"FindCoordinator": 0, "JoinGroup": 2, "Heartbeat": 301, "LeaveGroup": 1, "SyncGroup": 1, "SaslHandshake": 0, "ApiVersion": 1, "SaslAuthenticate":
0, "DescribeCluster": 0, "DescribeProducers": 0, "Unknown-62?": 0, "DescribeTransactions": 0, "ListTransactions": 0, "Unknown-69?": 0,
"Unknown-70?": 0, "GetTelemetrySubscriptions": 0, "PushTelemetry": 0, "Unknown-73?": 0 }, "toppars":{ } } }, "topics":{ "ProtobufListProtodelim": {
"topic":"ProtobufListProtodelim", "age":1968314, "metadata_age":168318, "batchsize": { "min":0, "max":0, "avg":0, "sum":0, "stddev": 0, "p50": 0,
"p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0, "cnt":0 }, "batchcnt": { "min":0, "max":0, "avg":0, "sum":0,
"stddev": 0, "p50": 0, "p75": 0, "p90": 0, "p95": 0, "p99": 0, "p99_99": 0, "outofrange": 0, "hdrsize": 0, "cnt":0 }, "partitions":{ "0": {
"partition":0, "broker":0, "leader":0, "desired":true, "unknown":false, "msgq_cnt":0, "msgq_bytes":0, "xmit_msgq_cnt":0, "xmit_msgq_bytes":0,
"fetchq_cnt":4816, "fetchq_size":9522794, "fetch_state":"active", "query_offset":-2, "next_offset":10, "app_offset":10, "stored_offset":9,
"stored_leader_epoch":-1, "commited_offset":-1001, "committed_offset":-1001, "committed_leader_epoch":-1, "eof_offset":-1001, "lo_offset":0,
"hi_offset":4168, "ls_offset":4168, "consumer_lag":-1, "consumer_lag_stored":4159, "leader_epoch":-1, "txmsgs":0, "txbytes":0, "rxmsgs":4168,
"rxbytes":9543544, "msgs": 4168, "rx_ver_drops": 0, "msgs_inflight": 0, "next_ack_seq": 0, "next_err_seq": 0, "acked_msgid": 0} , "-1": {
"partition":-1, "broker":-1, "leader":-1, "desired":false, "unknown":false, "msgq_cnt":0, "msgq_bytes":0, "xmit_msgq_cnt":0, "xmit_msgq_bytes":0,
"fetchq_cnt":0, "fetchq_size":0, "fetch_state":"none", "query_offset":-1001, "next_offset":0, "app_offset":-1001, "stored_offset":-1001,
"stored_leader_epoch":-1, "commited_offset":-1001, "committed_offset":-1001, "committed_leader_epoch":-1, "eof_offset":-1001, "lo_offset":-1001,
"hi_offset":-1001, "ls_offset":-1001, "consumer_lag":-1, "consumer_lag_stored":-1, "leader_epoch":-1, "txmsgs":0, "txbytes":0, "rxmsgs":0,
"rxbytes":0, "msgs": 0, "rx_ver_drops": 0, "msgs_inflight": 0, "next_ack_seq": 0, "next_err_seq": 0, "acked_msgid": 0} } } } , "cgrp": { "state":
"up", "stateage": 1968317, "join_state": "wait-unassign-call", "rebalance_age": 1067882, "rebalance_cnt": 2, "rebalance_reason":
"max.poll.interval.ms exceeded", "assignment_size": 0 }, "tx":2992, "tx_bytes":491395, "rx":2992, "rx_bytes":3156032, "txmsgs":0, "txmsg_bytes":0,
"rxmsgs":4168, "rxmsg_bytes":9543544}
```

```
d2775600408f :) select * from ProtobufListProtodelim;

SELECT *
FROM ProtobufListProtodelim

Query id: bb4d6ee1-f0db-4162-b4a5-589f22c31bc5


┌──────────────────Timestamp_Ns─┬─Hostname─────┬─Pop─┬─Label─┬─Tag─┬─Poll_Counter─┬─Record_Counter─┬─Function──────────────┬─Variable──────┬─Type──
──┬─Value─┐
1. │ 2025-04-07 18:04:21.616201472 │ c2c270b37cd9 │     │       │     │            0 │              0 │ Poller                │
pollingLoops │ count   │     1 │
2. │ 2025-04-07 18:04:21.616201472 │ c2c270b37cd9 │     │       │     │            0 │              1 │ Poller                │
ticker       │ count   │     1 │
3. │ 2025-04-07 18:04:21.616201472 │ c2c270b37cd9 │     │       │     │            0 │              2 │ Run                   │ start
             │ counter │     1 │
4. │ 2025-04-07 18:04:21.616201472 │ c2c270b37cd9 │     │       │     │            0 │              3 │ fetchPrometheusMetrics │ start
             │ count   │     1 │
5. │ 2025-04-07 18:04:21.616201472 │ c2c270b37cd9 │     │       │     │            0 │              4 │ performPoll           │ start
             │ count   │     1 │

└──────────────────────────────┴──────────────┴─────┴───────┴─────┴──────────────┴────────────────┴───────────────────────┴───────────────┴──────
──┴───────┘

5 rows in set. Elapsed: 0.006 sec.
```

Why do I think this should work?

So it doesn't work, but why not? I don't know. In this section, I'll demonstrate manually inserting data into the clickhouse table and putting it back in. I'll also prove that the way I'm encoding the protobuf is the same, because I can generate equivalent binary messages. I'll also prove that reading directly from the Kafka pipeline decodes correctly.

To set up for the test, let's drop the tables that aren't working. Please note that dropping the _kafka table takes a long time.

```
7392f49b6a57 :) show tables;

SHOW TABLES

Query id: ee3b87db-20cc-42d3-af7d-1cf6d4fc1c8c

    ┌─name──────────────────────────┐
1.  │ ProtobufListProtodelim        │
2.  │ ProtobufListProtodelim_kafka  │
3.  │ ProtobufListProtodelim_mv     │
    └───────────────────────────────┘

3 rows in set. Elapsed: 0.044 sec.

7392f49b6a57 :) DROP TABLE ProtobufListProtodelim_kafka

DROP TABLE ProtobufListProtodelim_kafka

Query id: 38271682-000b-441c-8e2c-e79c3411cbc1

Ok.

0 rows in set. Elapsed: 170.004 sec.

7392f49b6a57 :) DROP TABLE ProtobufListProtodelim_mv

DROP TABLE ProtobufListProtodelim_mv

Query id: c42cdacf-ff40-474f-b557-0720c25cd61b

Ok.

0 rows in set. Elapsed: 0.001 sec.

7392f49b6a57 :) show tables;

SHOW TABLES

Query id: f406c539-3f15-4819-b1a5-d7439deb9bf4

    ┌─name──────────────────┐
1.  │ ProtobufListProtodelim │
    └────────────────────────┘

1 row in set. Elapsed: 0.004 sec.
```

Prepare for experiment

Now, let's insert some rows in this table, so we can export them check we know what we are doing here. To do this, I've generate the insert statement here: https://github.com/randomizedcoder/gdp/blob/main/build/containers/clickhouse/initdb.d/sql/gdp/ProtobufListProtodelim_insert_into.sql

```
7392f49b6a57 :) INSERT INTO gdp.ProtobufListProtodelim (
          Timestamp_Ns,
          Hostname,
          Pop,
          Label,
          Tag,
          Poll_Counter,
          Record_Counter,
          Function,
          Variable,
          Type,
          Value
) VALUES
(
          toDateTime64('2025-04-05 10:00:00.0', 9, 'UTC'),
          'host1',
          'pop1',
          'label1',
          'tag1',
          100,
          1,
          'function1',
          'variable1',
          'type1',
          123.45
),
(
          toDateTime64('2025-04-05 11:00:00.0', 9, 'UTC'),
          'host2',
          'pop2',
          'label2',
          'tag2',
          200,
          2,
          'function2',
          'variable2',
          'type2',
          678.90
```

```
);

INSERT INTO gdp.ProtobufListProtodelim (Timestamp_Ns, Hostname, Pop, Label, Tag, Poll_Counter, Record_Counter, Function, Variable, Type, Value)
FORMAT Values

Query id: 2c19f28f-9e8c-49af-a1c6-224af333e83f

Ok.

2 rows in set. Elapsed: 0.046 sec.

7392f49b6a57 :) SELECT * FROM gdp.ProtobufListProtodelim LIMIT 10;

SELECT *
FROM gdp.ProtobufListProtodelim
LIMIT 10

Query id: e981a479-79a4-4244-8089-e6c4da0a31cb

   ┌──────────────Timestamp_Ns─┬─Hostname─┬─Pop──┬─Label──┬─Tag──┬─Poll_Counter─┬─Record_Counter─┬─Function──┬─Variable──┬─Type──┬─Value─┐
1. │ 2025-04-05 10:00:00.000000000 │ host1    │ pop1 │ label1 │ tag1 │          100 │              1 │ function1 │ variable1 │ type1 │
123.45 │
2. │ 2025-04-05 11:00:00.000000000 │ host2    │ pop2 │ label2 │ tag2 │          200 │              2 │ function2 │ variable2 │ type2 │
678.9 │
   └──────────────────────────┴──────────┴──────┴────────┴──────┴──────────────┴────────────────┴───────────┴───────────┴───────┴───────┘

2 rows in set. Elapsed: 0.008 sec.
```

Insert some test rows into the table
https://github.com/randomizedcoder/gdp/blob/main/build/containers/clickhouse/initdb.d/sql/gdp/ProtobufListProtodelim_insert_into.sql

Now we can export the rows as a ProtobufList type. This proves the schema registry is good, and gives us a binary payload we can look at, and we can put it back into the DB.

Let's export the rows. The SQL to do this is here:
https://github.com/randomizedcoder/gdp/blob/main/build/containers/clickhouse/initdb.d/sql/export_protobufListProtodelim.sql

```
7392f49b6a57 :) SELECT
  *
FROM gdp.ProtobufListProtodelim
LIMIT 2
INTO OUTFILE 'gdp.ProtobufListProtodelim.bin'
FORMAT ProtobufList
SETTINGS format_schema = '/var/lib/clickhouse/format_schemas/prometheus_protolist.proto:PromRecordCounter'


SELECT *
FROM gdp.ProtobufListProtodelim
LIMIT 2
INTO OUTFILE 'gdp.ProtobufListProtodelim.bin'
FORMAT ProtobufList
SETTINGS format_schema = '/var/lib/clickhouse/format_schemas/prometheus_protolist.proto:PromRecordCounter'

Query id: 95b682b8-0abd-4146-937a-68375907268c


2 rows in set. Elapsed: 0.003 sec.
```

```
7392f49b6a57 :) SELECT
  *
FROM gdp.ProtobufListProtodelim
LIMIT 2
INTO OUTFILE 'gdp.ProtobufListProtodelim.bin'
FORMAT ProtobufList
SETTINGS format_schema = '/var/lib/clickhouse/format_schemas/prometheus_protolist.proto:PromRecordCounter'


SELECT *
FROM gdp.ProtobufListProtodelim
LIMIT 2
INTO OUTFILE 'gdp.ProtobufListProtodelim.bin'
FORMAT ProtobufList
SETTINGS format_schema = '/var/lib/clickhouse/format_schemas/prometheus_protolist.proto:PromRecordCounter'

Query id: 95b682b8-0abd-4146-937a-68375907268c


2 rows in set. Elapsed: 0.003 sec.

7392f49b6a57 :) exit
Bye.
root@7392f49b6a57:/# ls -la | grep ProtobufListProtodelim
-rw-r--r--   1 root root      183 Apr  7 00:21 gdp.ProtobufListProtodelim.bin
```

Exporting the rows and checking the file

Now, let’s clear the rows in the table, and check we can insert this payload back in. (This is basically what the clickhouse test 02240 does.)

```
root@7392f49b6a57:/# clickhouse-client
ClickHouse client version 25.3.1.2703 (official build).
Connecting to localhost:9000 as user default.
Connected to ClickHouse server version 25.3.1.

Warnings:
 * Delay accounting is not enabled, OSIOWaitMicroseconds will not be gathered. You can enable it using `echo 1 > /proc/sys/kernel/task_delayacct`
or by using sysctl.

7392f49b6a57 :) use gdp

USE gdp

Query id: fe3e5d53-ae55-4300-8e7f-d9fa04d0e660

Ok.

0 rows in set. Elapsed: 0.001 sec.

7392f49b6a57 :) show tables

SHOW TABLES

Query id: bb57b036-81d0-45b2-9c64-1850e6c10ab2

   ┌─name──────────────────┐
1. │ ProtobufListProtodelim │
   └────────────────────────┘

1 row in set. Elapsed: 0.001 sec.

7392f49b6a57 :) TRUNCATE TABLE ProtobufListProtodelim;

TRUNCATE TABLE ProtobufListProtodelim

Query id: 69c41c66-4dce-4166-829f-710f1f2a3dcc

Ok.

0 rows in set. Elapsed: 0.002 sec.

7392f49b6a57 :) SELECT * FROM gdp.ProtobufListProtodelim;

SELECT *
FROM gdp.ProtobufListProtodelim

Query id: 99582205-ad14-4d52-bb48-17c6c9fd5386

Ok.

0 rows in set. Elapsed: 0.001 sec.
```

Clear the rows

Now let’s insert them back

The command to insert them back in is

```
clickhouse-client --query "INSERT INTO gdp.ProtobufListProtodelim SETTINGS
format_schema='/var/lib/clickhouse/format_schemas/prometheus_protolist.proto:PromRecordCounter' FORMAT
ProtobufList" < gdp.ProtobufListProtodelim.bin
```

```
root@7392f49b6a57:/# clickhouse-client --query "INSERT INTO gdp.ProtobufListProtodelim SETTINGS
format_schema='/var/lib/clickhouse/format_schemas/prometheus_protolist.proto:PromRecordCounter' FORMAT ProtobufList" <
gdp.ProtobufListProtodelim.bin
root@7392f49b6a57:/# clickhouse-client
ClickHouse client version 25.3.1.2703 (official build).
Connecting to localhost:9000 as user default.
Connected to ClickHouse server version 25.3.1.

Warnings:
 * Delay accounting is not enabled, OSIOWaitMicroseconds will not be gathered. You can enable it using `echo 1 > /proc/sys/kernel/task_delayacct`
or by using sysctl.

7392f49b6a57 :) USE gdp

USE gdp

Query id: 6799df4b-f0c9-4146-81eb-2076e7655d04
```

```
Ok.

0 rows in set. Elapsed: 0.002 sec.

7392f49b6a57 :) SELECT * FROM gdp.ProtobufListProtodelim;

SELECT *
FROM gdp.ProtobufListProtodelim

Query id: e70b8c75-f59e-4f9b-9423-f38e0828ed5c

   ┌──────────────Timestamp_Ns─┬─Hostname─┬─Pop──┬─Label──┬─Tag──┬─Poll_Counter─┬─Record_Counter─┬─Function──┬─Variable──┬─Type──┬─Value─┐
1. │ 2025-04-05 10:00:00.000000000 │ host1    │ pop1 │ label1 │ tag1 │          100 │              1 │ function1 │ variable1 │ type1 │
123.45 │
2. │ 2025-04-05 11:00:00.000000000 │ host2    │ pop2 │ label2 │ tag2 │          200 │              2 │ function2 │ variable2 │ type2 │
678.9 │
   └───────────────────────────────┴──────────┴──────┴────────┴──────┴──────────────┴────────────────┴───────────┴───────────┴───────┴───────┘

2 rows in set. Elapsed: 0.003 sec.
```

Insert the binary payload back into clickhouse

Please note that I suspect this works because the clickhouse-client might be doing the conversion, rather than the clickhouse-server. This is speculation.

Ok, so at this point, we know we can export and re-import, so protobuf schema stuff is working.

Can we read the payload we just exported?

Let's copy the payload out of the clickhouse docker container

```
[das@t:~/Downloads/gdp/cmd/build_binary_payloads]$ docker cp gdp-clickhouse-1:/gdp.ProtobufListProtodelim.bin ./
Successfully copied 2.05kB to /home/das/Downloads/gdp/cmd/build_binary_payloads/./

[das@t:~/Downloads/gdp/cmd/build_binary_payloads]$ ls -la
total 9968
drwxr-xr-x 2 das users        4096 Apr  6 17:33 .
drwxr-xr-x 9 das users        4096 Apr  5 08:43 ..
-rwxr-xr-x 1 das users 10159510 Apr  5 11:32 build_binary_payloads
-rw-r--r-- 1 das users        7669 Apr  6 17:19 build_binary_payloads.go
-rw-r--r-- 1 das users         183 Apr  6 17:21 gdp.ProtobufListProtodelim.bin
-rw-r--r-- 1 das users         183 Apr  5 11:30 gdp.ProtobufListProtodelim.bin_2025_04_05
-rw-r--r-- 1 das users         141 Apr  5 17:54 gdp.ProtobufListProtodelim.bin_2025_04_05_1755
-rw-r--r-- 1 das users          22 Apr  5 08:44 .gitignore
-rw-r--r-- 1 das users         879 Apr  5 08:44 Makefile
-rw-r--r-- 1 das users        4686 Apr  5 18:04 notes_2025_04_05

[das@t:~/Downloads/gdp/cmd/build_binary_payloads]$ date
Sun Apr  6 05:33:15 PM PDT 2025
```

Copy the payload out of the clickhouse docker, and into /gdp/cmd/build_binary_payload/
https://github.com/randomizedcoder/gdp/tree/main/cmd/build_binary_payloads

Now let's use this little tool to "read" the payload, and see if it matches.

```
[das@t:~/Downloads/gdp/cmd/build_binary_payloads]$ ./build_binary_payloads --help
Usage of ./build_binary_payloads:
  -d uint
        debug level (default 11)
  -f string
        file name (default "gdp.ProtobufListProtodelim.bin")
  -of string
        output folder (default "./")
  -r    read rows
  -w    write

[das@t:~/Downloads/gdp/cmd/build_binary_payloads]$ ./build_binary_payloads -r
2025/04/07 00:36:10.522275 build_binary_payloads.go:81: Reading file: ./gdp.ProtobufListProtodelim.bin
{"timestampNs":1743847200, "hostname":"host1", "pop":"pop1", "label":"label1", "tag":"tag1", "pollCounter":"100", "recordCounter":"1",
"function":"function1", "variable":"variable1", "type":"type1", "value":123.45}
{"timestampNs":1743850800, "hostname":"host2", "pop":"pop2", "label":"label2", "tag":"tag2", "pollCounter":"200", "recordCounter":"2",
"function":"function2", "variable":"variable2", "type":"type2", "value":678.9}
```

Decode the binary using build_binary_payloads = yay! The Records match! 🚀

Ok, great. What about testing our gdp marshalling function to check what we are generating matches the binary payload. To do this, I just hacked the code with hard coded values. See them here

https://github.com/randomizedcoder/gdp/blob/main/cmd/build_binary_payloads/build_binary_payloads.go#L207

```
[das@t:~/Downloads/gdp/cmd/build_binary_payloads]$ date
Sun Apr  6 05:50:03 PM PDT 2025

[das@t:~/Downloads/gdp/cmd/build_binary_payloads]$ ./build_binary_payloads -f gdp.ProtobufListProtodelim.bin_2025_04_06_05_50 -w
2025/04/07 00:50:40.214293 build_binary_payloads.go:201: Wrote file: ./gdp.ProtobufListProtodelim.bin_2025_04_06_05_50

[das@t:~/Downloads/gdp/cmd/build_binary_payloads]$ ls -la
total 9972
drwxr-xr-x 2 das users        4096 Apr  6 17:50 .
drwxr-xr-x 9 das users        4096 Apr  5 08:43 ..
-rwxr-xr-x 1 das users 10159510 Apr  5 11:32 build_binary_payloads
-rw-r--r-- 1 das users        7669 Apr  6 17:19 build_binary_payloads.go
-rw-r--r-- 1 das users         183 Apr  6 17:21 gdp.ProtobufListProtodelim.bin
-rw-r--r-- 1 das users         183 Apr  5 11:30 gdp.ProtobufListProtodelim.bin_2025_04_05
-rw-r--r-- 1 das users         141 Apr  5 17:54 gdp.ProtobufListProtodelim.bin_2025_04_05_1755
-rw-r--r-- 1 das users         183 Apr  6 17:50 gdp.ProtobufListProtodelim.bin_2025_04_06_05_50        <-- new file
-rw-r--r-- 1 das users          22 Apr  5 08:44 .gitignore
-rw-r--r-- 1 das users         879 Apr  5 08:44 Makefile
-rw-r--r-- 1 das users        4686 Apr  5 18:04 notes_2025_04_05
```

Generate the same binary using gdp marshal function

We already see the sizes match, but let's do a diff, just to be sure

```
[das@t:~/Downloads/gdp/cmd/build_binary_payloads]$ diff gdp.ProtobufListProtodelim.bin
gdp.ProtobufListProtodelim.bin_2025_04_06_05_50

[das@t:~/Downloads/gdp/cmd/build_binary_payloads]$ hexdump gdp.ProtobufListProtodelim.bin
0000000 01b5 580a 0051 0000 3fc8 d9fc a241 0501
0000010 6f68 7473 f231 0401 6f70 3170 0392 6c06
0000020 6261 6c65 e231 0403 6174 3167 04b0 8064
0000030 0105 05d2 6609 6e75 7463 6f69 316e 06a2
0000040 7609 7261 6169 6c62 3165 06f2 7405 7079
0000050 3165 07c1 cccd cccc dccc 405e 590a 0051
0000060 0000 434c d9fc a241 0501 6f68 7473 f232
0000070 0401 6f70 3270 0392 6c06 6261 6c65 e232
0000080 0403 6174 3267 04b0 01c8 0580 d202 0905
0000090 7566 636e 6974 6e6f a232 0906 6176 6972
00000a0 6261 656c f232 0506 7974 6570 c132 3307
00000b0 3333 3333 8537 0040
00000b7

[das@t:~/Downloads/gdp/cmd/build_binary_payloads]$ hexdump gdp.ProtobufListProtodelim.bin_2025_04_06_05_50
0000000 01b5 580a 0051 0000 3fc8 d9fc a241 0501
0000010 6f68 7473 f231 0401 6f70 3170 0392 6c06
0000020 6261 6c65 e231 0403 6174 3167 04b0 8064
0000030 0105 05d2 6609 6e75 7463 6f69 316e 06a2
0000040 7609 7261 6169 6c62 3165 06f2 7405 7079
0000050 3165 07c1 cccd cccc dccc 405e 590a 0051
0000060 0000 434c d9fc a241 0501 6f68 7473 f232
0000070 0401 6f70 3270 0392 6c06 6261 6c65 e232
0000080 0403 6174 3267 04b0 01c8 0580 d202 0905
0000090 7566 636e 6974 6e6f a232 0906 6176 6972
00000a0 6261 656c f232 0506 7974 6570 c132 3307
00000b0 3333 3333 8537 0040
00000b7
```

The files are a perfect match 😕 So why doesn't our payload work when we send it via Kafka? 😕

Another reason, why do I think this should work?

There's one more test we can do. There is the small tool that simply reads from Kafka, and can decode the payload based on the cli arguments. I listed this else where in the doc, and all the code is in this folder: https://github.com/randomizedcoder/gdp/blob/main/cmd/gdp_kafka_client/gdp_kafka_client.go

Using this tool, we can connect to the Kafka topic ProtobufListProtodelim, and tell the little tool to decode what it gets as the Envelope, with no kafka header, and with the length delimiting.

The cli options are like this:

```
./gdp_kafka_client -topic ProtobufListProtodelim -pb envelope -k=false -delim=true

Or do

make envelope_delim
```

If you run this command more than once, Kafka will have remembered where the consumer group is up too, so you can pass another group name using the “--group another_group_name”

```
[das@t:~/Downloads/gdp/cmd/gdp_kafka_client]$ ./gdp_kafka_client -topic ProtobufListProtodelim -pb envelope -k=false -delim=true -group
2025_05_06_1800 2>&1 | more
2025/04/07 01:00:14.106826 gdp_kafka_client.go:55: Parsed flags:
2025/04/07 01:00:14.106895 gdp_kafka_client.go:56:   Debug Level: 11
2025/04/07 01:00:14.106908 gdp_kafka_client.go:57:   Topic: ProtobufListProtodelim
2025/04/07 01:00:14.106918 gdp_kafka_client.go:58:   Consumer Group: 2025_05_06_1800
2025/04/07 01:00:14.106927 gdp_kafka_client.go:59:   Protobuf Type: envelope
2025/04/07 01:00:14.106936 gdp_kafka_client.go:60:   Kafka Header: false
2025/04/07 01:00:14.106946 gdp_kafka_client.go:61:   Length-Delimited: true
2025/04/07 01:00:14.107261 gdp_kafka_client.go:92: kgo.NewClient complete
2025/04/07 01:00:14.123313 gdp_kafka_client.go:148: processRecordInner before UnmarshalFrom
2025/04/07 01:00:14.123467 gdp_kafka_client.go:191: Processed message:
{"rows":[{"timestampNs":1743892572.1020591,"hostname":"1d93eab32382","function":"Poller","variable":"pollingLoops","type":"count","value":1},{"time
stampNs":1743892572.1020591,"hostname":"1d93eab32382","recordCounter":"1","function":"Poller","variab
le":"ticker","type":"count","value":1},{"timestampNs":1743892572.1020591,"hostname":"1d93eab32382","recordCounter":"2","function":"Run","variable":
"start","type":"counter","value":1},{"timestampNs":1743892572.1020591,"hostname":"1d93eab32382","recordCounter":"3","function":"fetchPrometheusMetr
ics","variable":"start","
type":"count","value":1},{"timestampNs":1743892572.1020591,"hostname":"1d93eab32382","recordCounter":"4","function":"performPoll","variable":"start
","type":"count","value":1}]}
2025/04/07 01:00:14.123476 gdp_kafka_client.go:148: processRecordInner before UnmarshalFrom
2025/04/07 01:00:14.123559 gdp_kafka_client.go:191: Processed message:
{"rows":[{"timestampNs":1743892632.108121,"hostname":"1d93eab32382","pollCounter":"1","function":"Poller","variable":"pollingLoops","type":"count",
"value":2},{"timestampNs":1743892632.108121,"hostname":"1d93eab32382","pollCounter":"1","recordCounte
r":"1","function":"Poller","variable":"ticker","type":"count","value":2},{"timestampNs":1743892632.108121,"hostname":"1d93eab32382","pollCounter":"
1","recordCounter":"2","function":"Run","variable":"start","type":"counter","value":1},{"timestampNs":1743892632.108121,"hostname":"1d93eab32382","
pollCounter":"1","recordC
ounter":"3","function":"destKafka","variable":"ProtobufList","type":"count","value":1},{"timestampNs":1743892632.108121,"hostname":"1d93eab32382","
pollCounter":"1","recordCounter":"4","function":"destKafka","variable":"ProtobufListKafka","type":"count","value":1},{"timestampNs":1743892632.1081
21,"hostname":"1d93eab323
82","pollCounter":"1","recordCounter":"5","function":"destKafka","variable":"ProtobufListKafkaProtodelim","type":"count","value":1},{"timestampNs":
1743892632.108121,"hostname":"1d93eab32382","pollCounter":"1","recordCounter":"6","function":"destKafka","variable":"ProtobufListProtodelim","type"
:"count","value":1},{"tim
estampNs":1743892632.108121,"hostname":"1d93eab32382","pollCounter":"1","recordCounter":"7","function":"destKafka","variable":"ProtobufSingle","typ
e":"count","value":5},{"timestampNs":1743892632.108121,"hostname":"1d93eab32382","pollCounter":"1","recordCounter":"8","function":"destKafka","vari
able":"ProtobufSingleKafk
a","type":"count","value":5},{"timestampNs":1743892632.108121,"hostname":"1d93eab32382","pollCounter":"1","recordCounter":"9","function":"destKafka
","variable":"ProtobufSingleKafkaProtodelim","type":"count","value":5},{"timestampNs":1743892632.108121,"hostname":"1d93eab32382","pollCounter":"1"
,"recordCounter":"10","fu
nction":"destKafka","variable":"ProtobufSingleProtodelim","type":"count","value":5},{"timestampNs":1743892632.108121,"hostname":"1d93eab32382","pol
lCounter":"1","recordCounter":"11","function":"destKafka","variable":"start","type":"count","value":24},{"timestampNs":1743892632.108121,"hostname"
:"1d93eab32382","pollCoun
ter":"1","recordCounter":"12","function":"destKafka","variable":"ProtobufList","type":"n","value":345},{"timestampNs":1743892632.108121,"hostname":
"1d93eab32382","pollCounter":"1","recordCounter":"13","function":"destKafka","variable":"ProtobufListKafka","type":"n","value":351},{"timestampNs":
1743892632.108121,"hostna
me":"1d93eab32382","pollCounter":"1","recordCounter":"14","function":"destKafka","variable":"ProtobufListKafkaProtodelim","type":"n","value":353},{
"timestampNs":1743892632.108121,"hostname":"1d93eab32382","pollCounter":"1","recordCounter":"15","function":"destKafka","variable":"ProtobufListPro
todelim","type":"n","valu
e":347},{"timestampNs":1743892632.108121,"hostname":"1d93eab32382","pollCounter":"1","recordCounter":"16","function":"destKafka","variable":"Protob
ufSingle","type":"n","value":335},{"timestampNs":1743892632.108121,"hostname":"1d93eab32382","pollCounter":"1","recordCounter":"17","function":"des
tKafka","variable":"Proto
bufSingleKafka","type":"n","value":365},{"timestampNs":1743892632.108121,"hostname":"1d93eab32382","pollCounter":"1","recordCounter":"18","function
":"destKafka","variable":"ProtobufSingleKafkaProtodelim","type":"n","value":370},{"timestampNs":1743892632.108121,"hostname":"1d93eab32382","pollCo
unter":"1","recordCounter
":"19","function":"destKafka","variable":"ProtobufSingleProtodelim","type":"n","value":340},{"timestampNs":1743892632.108121,"hostname":"1d93eab323
82","pollCounter":"1","recordCounter":"20","function":"fetchPrometheusMetrics","variable":"bodyBytes","type":"count","value":8640},{"timestampNs":1
743892632.108121,"hostnam
e":"1d93eab32382","pollCounter":"1","recordCounter":"21","function":"fetchPrometheusMetrics","variable":"firstShot","type":"count","value":1},{"tim
estampNs":1743892632.108121,"hostname":"1d93eab32382","pollCounter":"1","recordCounter":"22","function":"fetchPrometheusMetrics","variable":"start"
,"type":"count","value":2
},{"timestampNs":1743892632.108121,"hostname":"1d93eab32382","pollCounter":"1","recordCounter":"23","function":"fetchPrometheusMetrics","variable":
"success","type":"count","value":1},{"timestampNs":1743892632.108121,"hostname":"1d93eab32382","pollCounter":"1","recordCounter":"24","function":"f
ilterCounts","variable":"
counts","type":"count","value":5},{"timestampNs":1743892632.108121,"hostname":"1d93eab32382","pollCounter":"1","recordCounter":"25","function":"fil
terCounts","variable":"filtered","type":"count","value":125},{"timestampNs":1743892632.108121,"hostname":"1d93eab32382","pollCounter":"1","recordCo
unter":"26","function":"p
erformPoll","variable":"start","type":"count","value":2},{"timestampNs":1743892632.108121,"hostname":"1d93eab32382","pollCounter":"1","recordCounte
r":"27","function":"sendEnvelopeWithMarshalConfig","variable":"start","type":"count","value":8}]}
2025/04/07 01:00:14.123568 gdp_kafka_client.go:148: processRecordInner before UnmarshalFrom
2025/04/07 01:00:14.123660 gdp_kafka_client.go:191: Processed message:
{"rows":[{"timestampNs":1743892692.1188624,"hostname":"1d93eab32382","pollCounter":"2","function":"Poller","variable":"pollingLoops","type":"count"
,"value":3},{"timestampNs":1743892692.1188624,"hostname":"1d93eab32382","pollCounter":"2","recordCoun
ter":"1","function":"Poller","variable":"ticker","type":"count","value":3},{"timestampNs":1743892692.1188624,"hostname":"1d93eab32382","pollCounter
":"2","recordCounter":"2","function":"Run","variable":"start","type":"counter","value":1},{"timestampNs":1743892692.1188624,"hostname":"1d93eab3238
2","pollCounter":"2","rec
ordCounter":"3","function":"destKafka","variable":"ProtobufList","type":"count","value":2},{"timestampNs":1743892692.1188624,"hostname":"1d93eab323
82","pollCounter":"2","recordCounter":"4","function":"destKafka","variable":"ProtobufListKafka","type":"count","value":2},{"timestampNs":1743892692
.1188624,"hostname":"1d93
eab32382","pollCounter":"2","recordCounter":"5","function":"destKafka","variable":"ProtobufListKafkaProtodelim","type":"count","value":2},{"timesta
mpNs":1743892692.1188624,"hostname":"1d93eab32382","pollCounter":"2","recordCounter":"6","function":"destKafka","variable":"ProtobufListProtodelim"
,"type":"count","value":2
```

},{"timestampNs":1743892692.1188624,"hostname":"1d93eab32382","pollCounter":"2","recordCounter":"7","function":"destKafka","variable":"ProtobufSing
le","type":"count","value":33},{"timestampNs":1743892692.1188624,"hostname":"1d93eab32382","pollCounter":"2","recordCounter":"8","function":"destKa
fka","variable":"Protobuf
SingleKafka","type":"count","value":33},{"timestampNs":1743892692.1188624,"hostname":"1d93eab32382","pollCounter":"2","recordCounter":"9","function
":"destKafka","variable":"ProtobufSingleKafkaProtodelim","type":"count","value":33},{"timestampNs":1743892692.1188624,"hostname":"1d93eab32382","po
llCounter":"2","recordCou
nter":"10","function":"destKafka","variable":"ProtobufSingleProtodelim","type":"count","value":33},{"timestampNs":1743892692.1188624,"hostname":"1d
93eab32382","pollCounter":"2","recordCounter":"11","function":"destKafka","variable":"start","type":"count","value":140},{"timestampNs":1743892692.
1188624,"hostname":"1d93e
ab32382","pollCounter":"2","recordCounter":"12","function":"destKafka","variable":"ProtobufList","type":"n","value":2610},{"timestampNs":1743892692
.1188624,"hostname":"1d93eab32382","pollCounter":"2","recordCounter":"13","function":"destKafka","variable":"ProtobufListKafka","type":"n","value":
2622},{"timestampNs":1743
892692.1188624,"hostname":"1d93eab32382","pollCounter":"2","recordCounter":"14","function":"destKafka","variable":"ProtobufListKafkaProtodelim","ty
pe":"n","value":2626},{"timestampNs":1743892692.1188624,"hostname":"1d93eab32382","pollCounter":"2","recordCounter":"15","function":"destKafka","va
riable":"ProtobufListProt
odelim","type":"n","value":2614},{"timestampNs":1743892692.1188624,"hostname":"1d93eab32382","pollCounter":"2","recordCounter":"16","function":"des
tKafka","variable":"ProtobufSingle","type":"n","value":2544},{"timestampNs":1743892692.1188624,"hostname":"1d93eab32382","pollCounter":"2","recordC
ounter":"17","function":"
destKafka","variable":"ProtobufSingleKafka","type":"n","value":2742},{"timestampNs":1743892692.1188624,"hostname":"1d93eab32382","pollCounter":"2",
"recordCounter":"18","function":"destKafka","variable":"ProtobufSingleKafkaProtodelim","type":"n","value":2775},{"timestampNs":1743892692.1188624,"
hostname":"1d93eab32382",
"pollCounter":"2","recordCounter":"19","function":"destKafka","variable":"ProtobufSingleProtodelim","type":"n","value":2577},{"timestampNs":1743892
692.1188624,"hostname":"1d93eab32382","pollCounter":"2","recordCounter":"20","function":"fetchPrometheusMetrics","variable":"bodyBytes","type":"cou
nt","value":25298},{"time
stampNs":1743892692.1188624,"hostname":"1d93eab32382","pollCounter":"2","recordCounter":"21","function":"fetchPrometheusMetrics","variable":"firstS
hot","type":"count","value":2},{"timestampNs":1743892692.1188624,"hostname":"1d93eab32382","pollCounter":"2","recordCounter":"22","function":"fetch
PrometheusMetrics","varia
ble":"start","type":"count","value":3},{"timestampNs":1743892692.1188624,"hostname":"1d93eab32382","pollCounter":"2","recordCounter":"23","function
":"fetchPrometheusMetrics","variable":"success","type":"count","value":2},{"timestampNs":1743892692.1188624,"hostname":"1d93eab32382","pollCounter
":"2","recordCounter":"24
","function":"filterCounts","variable":"counts","type":"count","value":33},{"timestampNs":1743892692.1188624,"hostname":"1d93eab32382","pollCounter"
:"2","recordCounter":"25","function":"filterCounts","variable":"filtered","type":"count","value":312},{"timestampNs":1743892692.1188624,"hostname":
"1d93eab32382","pollCount
er":"2","recordCounter":"26","function":"performPoll","variable":"start","type":"count","value":3},{"timestampNs":1743892692.1188624,"hostname":"1d
93eab32382","pollCounter":"2","recordCounter":"27","function":"sendEnvelopeWithMarshalConfig","variable":"start","type":"count","value":16}]}
…
2025/04/07 01:00:49.936435 gdp_kafka_client.go:191: Processed message:
{"rows":[{"timestampNs":1743987570.2238436,"hostname":"1d93eab32382","pollCounter":"1583","function":"Poller","variable":"pollingLoops","type":"cou
nt","value":1584},{"timestampNs":1743987570.2238436,"hostname":"1d93eab32382","pollCounter":"1583","recordCounter":"1","function":"Poller","variabl
e":"ticker","type":"count","value":1584},{"timestampNs":1743987570.2238436,"hostname":"1d93eab32382","pollCounter":"1583","recordCounter":"2","funct
ion":"Run","variable":"start","type":"counter","value":1},{"timestampNs":1743987570.2238436,"hostname":"1d93eab32382","pollCounter":"1583","record
Counter":"3","function":"destKafka","variable":"ProtobufList","type":"count","value":1583},{"timestampNs":1743987570.2238436,"hostname":"1d93eab323
82","pollCounter":"1583","recordCounter":"4","function":"destKafka","variable":"ProtobufListKafka","type":"count","value":1583},{"timestampNs":1743
987570.2238436,"hostname":"1d93eab32382","pollCounter":"1583","recordCounter":"5","function":"destKafka","variable":"ProtobufListKafkaProtodelim","
type":"count","value":1583},{"timestampNs":1743987570.2238436,"hostname":"1d93eab32382","pollCounter":"1583","recordCounter":"6","function":"destKa
fka","variable":"ProtobufListProtodelim","type":"count","value":1583},{"timestampNs":1743987570.2238436,"hostname":"1d93eab32382","pollCounter":"15
83","recordCounter":"7","function":"destKafka","variable":"ProtobufSingle","type":"count","value":44301},{"timestampNs":1743987570.2238436,"hostnam
e":"1d93eab32382","pollCounter":"1583","recordCounter":"8","function":"destKafka","variable":"ProtobufSingleKafka","type":"count","value":44301},{"
timestampNs":1743987570.2238436,"hostname":"1d93eab32382","pollCounter":"1583","recordCounter":"9","function":"destKafka","variable":"ProtobufSingl
eKafkaProtodelim","type":"count","value":44301},{"timestampNs":1743987570.2238436,"hostname":"1d93eab32382","pollCounter":"1583","recordCounter":"1
0","function":"destKafka","variable":"ProtobufSingleProtodelim","type":"count","value":44301},{"timestampNs":1743987570.2238436,"hostname":"1d93eab
32382","pollCounter":"1583","recordCounter":"11","function":"destKafka","variable":"start","type":"count","value":183536},{"timestampNs":1743987570
.2238436,"hostname":"1d93eab32382","pollCounter":"1583","recordCounter":"12","function":"destKafka","variable":"ProtobufList","type":"n","value":36
24315},{"timestampNs":1743987570.2238436,"hostname":"1d93eab32382","pollCounter":"1583","recordCounter":"13","function":"destKafka","variable":"Pro
tobufListKafka","type":"n","value":3633813},{"timestampNs":1743987570.2238436,"hostname":"1d93eab32382","pollCounter":"1583","recordCounter":"14","
function":"destKafka","variable":"ProtobufListKafkaProtodelim","type":"n","value":3636979},{"timestampNs":1743987570.2238436,"hostname":"1d93eab323
82","pollCounter":"1583","recordCounter":"15","function":"destKafka","variable":"ProtobufListProtodelim","type":"n","value":3627481},{"timestampNs"
:1743987570.2238436,"hostname":"1d93eab32382","pollCounter":"1583","recordCounter":"16","function":"destKafka","variable":"ProtobufSingle","type":"
n","value":3535713},{"timestampNs":1743987570.2238436,"hostname":"1d93eab32382","pollCounter":"1583","recordCounter":"17","function":"destKafka","v
ariable":"ProtobufSingleKafka","type":"n","value":3801519},{"timestampNs":1743987570.2238436,"hostname":"1d93eab32382","pollCounter":"1583","record
Counter":"18","function":"destKafka","variable":"ProtobufSingleKafkaProtodelim","type":"n","value":3845820},{"timestampNs":1743987570.2238436,"host
name":"1d93eab32382","pollCounter":"1583","recordCounter":"19","function":"destKafka","variable":"ProtobufSingleProtodelim","type":"n","value":3580
014},{"timestampNs":1743987570.2238436,"hostname":"1d93eab32382","pollCounter":"1583","recordCounter":"20","function":"fetchPrometheusMetrics","var
iable":"bodyBytes","type":"count","value":26782140},{"timestampNs":1743987570.2238436,"hostname":"1d93eab32382","pollCounter":"1583","recordCounter
":"21","function":"fetchPrometheusMetrics","variable":"firstShot","type":"count","value":1583},{"timestampNs":1743987570.2238436,"hostname":"1d93ea
b32382","pollCounter":"1583","recordCounter":"22","function":"fetchPrometheusMetrics","variable":"start","type":"count","value":1584},{"timestampNs
":1743987570.2238436,"hostname":"1d93eab32382","pollCounter":"1583","recordCounter":"23","function":"fetchPrometheusMetrics","variable":"success","
type":"count","value":1583},{"timestampNs":1743987570.2238436,"hostname":"1d93eab32382","pollCounter":"1583","recordCounter":"24","function":"filte
rCounts","variable":"counts","type":"count","value":44301},{"timestampNs":1743987570.2238436,"hostname":"1d93eab32382","pollCounter":"1583","record
Counter":"25","function":"filterCounts","variable":"filtered","type":"count","value":295959},{"timestampNs":1743987570.2238436,"hostname":"1d93eab3
2382","pollCounter":"1583","recordCounter":"26","function":"performPoll","variable":"start","type":"count","value":1584},{"timestampNs":1743987570.
2238436,"hostname":"1d93eab32382","pollCounter":"1583","recordCounter":"27","function":"sendEnvelopeWithMarshalConfig","variable":"start","type":"c
ount","value":12664}]}
2025/04/07 01:00:49.936450 gdp_kafka_client.go:148: processRecordInner before UnmarshalFrom
2025/04/07 01:00:49.936516 gdp_kafka_client.go:191: Processed message:
{"rows":[{"timestampNs":1743987630.2300646,"hostname":"1d93eab32382","pollCounter":"1584","function":"Poller","variable":"pollingLoops","type":"cou
nt","value":1585},{"timestampNs":1743987630.2300646,"hostname":"1d93eab32382","pollCounter":"1584","recordCounter":"1","function":"Poller","variabl
e":"ticker","type":"count","value":1585},{"timestampNs":1743987630.2300646,"hostname":"1d93eab32382","pollCounter":"1584","recordCounter":"2","funct
ion":"Run","variable":"start","type":"counter","value":1},{"timestampNs":1743987630.2300646,"hostname":"1d93eab32382","pollCounter":"1584","record
Counter":"3","function":"destKafka","variable":"ProtobufList","type":"count","value":1584},{"timestampNs":1743987630.2300646,"hostname":"1d93eab323
82","pollCounter":"1584","recordCounter":"4","function":"destKafka","variable":"ProtobufListKafka","type":"count","value":1584},{"timestampNs":1743
987630.2300646,"hostname":"1d93eab32382","pollCounter":"1584","recordCounter":"5","function":"destKafka","variable":"ProtobufListKafkaProtodelim","
type":"count","value":1584},{"timestampNs":1743987630.2300646,"hostname":"1d93eab32382","pollCounter":"1584","recordCounter":"6","function":"destKa
fka","variable":"ProtobufListProtodelim","type":"count","value":1584},{"timestampNs":1743987630.2300646,"hostname":"1d93eab32382","pollCounter":"15
84","recordCounter":"7","function":"destKafka","variable":"ProtobufSingle","type":"count","value":44329},{"timestampNs":1743987630.2300646,"hostnam
e":"1d93eab32382","pollCounter":"1584","recordCounter":"8","function":"destKafka","variable":"ProtobufSingleKafka","type":"count","value":44329},{"
timestampNs":1743987630.2300646,"hostname":"1d93eab32382","pollCounter":"1584","recordCounter":"9","function":"destKafka","variable":"ProtobufSingl
eKafkaProtodelim","type":"count","value":44329},{"timestampNs":1743987630.2300646,"hostname":"1d93eab32382","pollCounter":"1584","recordCounter":"1
0","function":"destKafka","variable":"ProtobufSingleProtodelim","type":"count","value":44329},{"timestampNs":1743987630.2300646,"hostname":"1d93eab
32382","pollCounter":"1584","recordCounter":"11","function":"destKafka","variable":"start","type":"count","value":183652},{"timestampNs":1743987630
.2300646,"hostname":"1d93eab32382","pollCounter":"1584","recordCounter":"12","function":"destKafka","variable":"ProtobufList","type":"n","value":36
26608},{"timestampNs":1743987630.2300646,"hostname":"1d93eab32382","pollCounter":"1584","recordCounter":"13","function":"destKafka","variable":"Pro
tobufListKafka","type":"n","value":3636112},{"timestampNs":1743987630.2300646,"hostname":"1d93eab32382","pollCounter":"1584","recordCounter":"14","
function":"destKafka","variable":"ProtobufListKafkaProtodelim","type":"n","value":3639280},{"timestampNs":1743987630.2300646,"hostname":"1d93eab323
82","pollCounter":"1584","recordCounter":"15","function":"destKafka","variable":"ProtobufListProtodelim","type":"n","value":3629776},{"timestampNs"
:1743987630.2300646,"hostname":"1d93eab32382","pollCounter":"1584","recordCounter":"16","function":"destKafka","variable":"ProtobufSingle","type":"
n","value":3537950},{"timestampNs":1743987630.2300646,"hostname":"1d93eab32382","pollCounter":"1584","recordCounter":"17","function":"destKafka","v
ariable":"ProtobufSingleKafka","type":"n","value":3803924},{"timestampNs":1743987630.2300646,"hostname":"1d93eab32382","pollCounter":"1584","record
Counter":"18","function":"destKafka","variable":"ProtobufSingleKafkaProtodelim","type":"n","value":3848253},{"timestampNs":1743987630.2300646,"host
name":"1d93eab32382","pollCounter":"1584","recordCounter":"19","function":"destKafka","variable":"ProtobufSingleProtodelim","type":"n","value":3582
279},{"timestampNs":1743987630.2300646,"hostname":"1d93eab32382","pollCounter":"1584","recordCounter":"20","function":"fetchPrometheusMetrics","var
iable":"bodyBytes","type":"count","value":26799092},{"timestampNs":1743987630.2300646,"hostname":"1d93eab32382","pollCounter":"1584","recordCounter
":"21","function":"fetchPrometheusMetrics","variable":"firstShot","type":"count","value":1584},{"timestampNs":1743987630.2300646,"hostname":"1d93ea
b32382","pollCounter":"1584","recordCounter":"22","function":"fetchPrometheusMetrics","variable":"start","type":"count","value":1585},{"timestampNs

```
":1743987630.2300646,"hostname":"1d93eab32382","pollCounter":"1584","recordCounter":"23","function":"fetchPrometheusMetrics","variable":"success","
type":"count","value":1584},{"timestampNs":1743987630.2300646,"hostname":"1d93eab32382","pollCounter":"1584","recordCounter":"24","function":"filte
rCounts","variable":"counts","type":"count","value":44329},{"timestampNs":1743987630.2300646,"hostname":"1d93eab32382","pollCounter":"1584","record
Counter":"25","function":"filterCounts","variable":"filtered","type":"count","value":296146},{"timestampNs":1743987630.2300646,"hostname":"1d93eab3
2382","pollCounter":"1584","recordCounter":"26","function":"performPoll","variable":"start","type":"count","value":1585},{"timestampNs":1743987630.
2300646,"hostname":"1d93eab32382","pollCounter":"1584","recordCounter":"27","function":"sendEnvelopeWithMarshalConfig","variable":"start","type":"c
ount","value":12672}]}
^C2025/04/07 01:00:51.211019 gdp_kafka_client.go:202: Signal caught, closing application
2025/04/07 01:00:51.211044 gdp_kafka_client.go:205: Signal caught, cancel() called, and sleeping to allow goroutines to close
2025/04/07 01:00:51.211049 gdp_kafka_client.go:211: Sleep complete, goodbye! exit(0)
```


gdp_kafka_client also can demonstrate that what's written into Kafka is ok

So this leads me to suspect that using Kafka to ingest ProtobufList doesn't work. Maybe I'm doing something silly?

Github issues filed

Issue:
https://github.com/ClickHouse/ClickHouse/issues/78746

Docs issue:
https://github.com/ClickHouse/ClickHouse/issues/74769

Franz-go/Redpanda issue
https://github.com/twmb/franz-go/issues/953

Another way to reproduce this issue - "minimally"

The challenge here is that we need a Kafka system, such that we can push some messages into Kafka, and then let Clickhouse consume them. That's obviously a little tricky, and is probably why the Clickhouse tests don't have any Kafka tests, and probably why this bug exists, because it's untested.

Assuming a Kafka system is available, it is easy to run gdp to stream the data into the Kafka system. Please keep in mind that this takes the argument of a single broker, and of course the kafka system will pass back a set of machines for the client to connect to.

To do this, change to the ./cmd/gdp directory, run "make" to compile gdp, and then run gdp passing the "dest" argument, as shown here:
https://github.com/randomizedcoder/gdp/tree/main/cmd/gdp

```
[das@t:~/Downloads/gdp]$ cd cmd/gdp

[das@t:~/Downloads/gdp/cmd/gdp]$ make
[ -f gdp ] && rm -rf ./gdp || true
go build -ldflags \
         "-X main.commit=5a08527 -X main.date=2025-04-07-14:56 -X main.version=1.0.0" \
         -o ./gdp \
         ./gdp.go

[das@t:~/Downloads/gdp/cmd/gdp]$ ./gdp --help
Usage of ./gdp:
  -conf
        show config
  -d uint
        debug level (default 111)
  -dest string
        kafka:127.0.0.1:9092, udp:127.0.0.1:13000, or nsq:127.0.0.1:4150 (default "kafka:redpanda-0:9092")
  -destWriteFiles uint
        Write out the marshalled data to destWriteFiles number of files ( for debugging only )
  -frequency duration
        Poll frequency (default 1m0s)
  -goMaxProcs uint
        goMaxProcs = https://golang.org/pkg/runtime/#GOMAXPROCS (default 4)
```

```
  -kafkaSchemaUrl string
        kafka schema registry URL (default "http://redpanda-0:18081")
  -kd uint
        kafka debug level (default 1)
  -label string
        label applied to the protobuf
  -maxLoops uint
        Maximum number of loops, or zero (0) for forever
  -produceTimeout duration
        Kafka produce timeout (context.WithTimeout)
  -profile.mode string
        enable profiling mode, one of [cpu, mem, mutex, block]
  -promListProtoFile string
        promListProtoFileCst for registering with the schema registry (default "/prometheus_protolist.proto")
  -promListen string
        Prometheus http listening socket (default ":8888")
  -promPath string
        Prometheus http path (default "/metrics")
  -promProtoFile string
        promProtoFile for registering with the schema registry (default "/prometheus.proto")
  -tag string
        label applied to the protobuf
  -timeout duration
        Poll timeout per name space (default 5s)
  -v    show version

[das@t:~/Downloads/gdp/cmd/gdp]$  ./gdp -dest kafka:redpanda-0:9092
```

How to run gdp passing the Kafka broker name

To do this starting completely from a fresh situation:

```
git clone https://github.com/randomizedcoder/gdp
cd gdp/cmd/gdp
make
./gdp -dest kafka:redpanda-0:9092
```

Run gdp as a standalone binary, and specify the Kafka broker

This assumes you have go installed: https://go.dev/doc/install

Then once the data is streaming into Kafka create the Clickhouse tables using these. The only change required will be to change the kafka broker name to suit.
https://github.com/randomizedcoder/gdp/blob/main/build/containers/clickhouse/initdb.d/sql/gdp/ProtobufListProtodelim.sql
https://github.com/randomizedcoder/gdp/blob/main/build/containers/clickhouse/initdb.d/sql/gdp/ProtobufListProtodelim_kafka.sql
https://github.com/randomizedcoder/gdp/blob/main/build/containers/clickhouse/initdb.d/sql/gdp/ProtobufListProtodelim_mv.sql

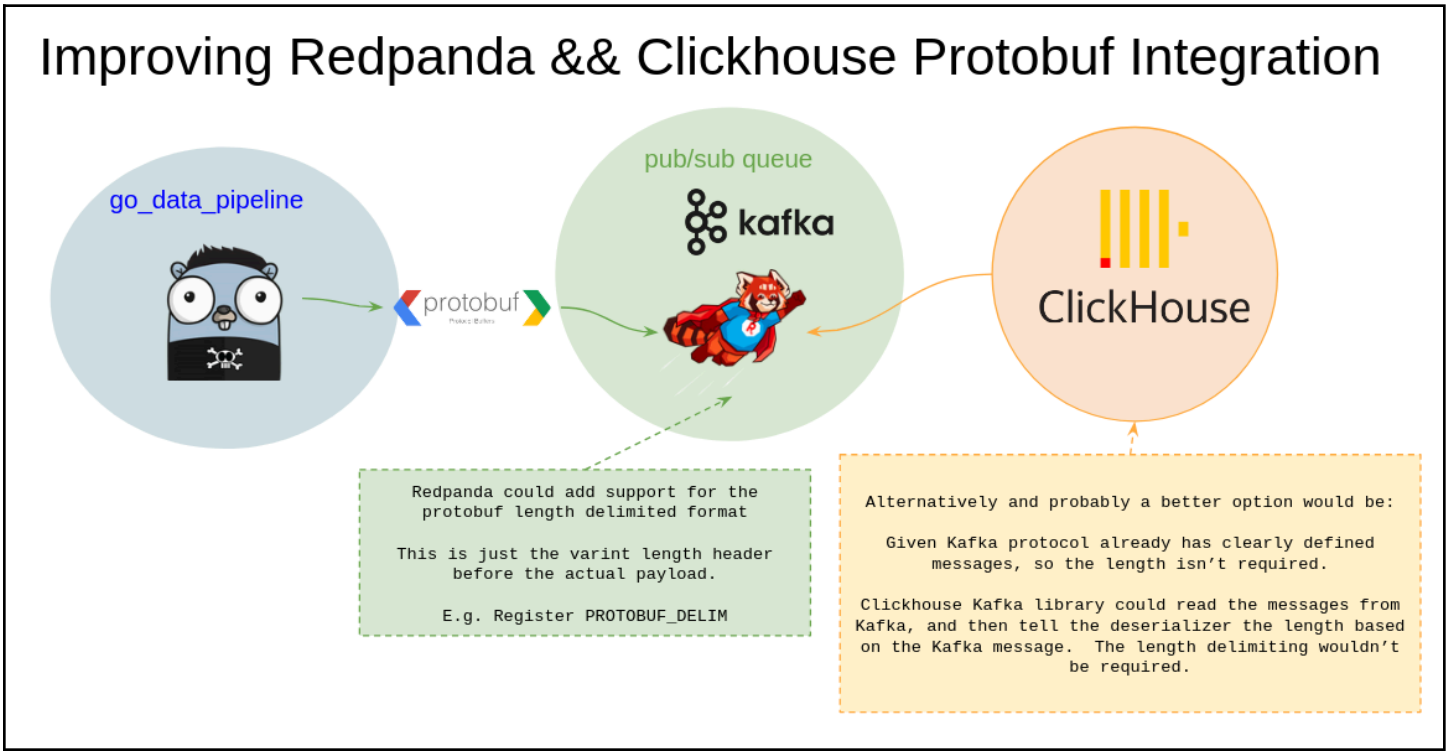
```
CREATE TABLE IF NOT EXISTS gdp.ProtobufListProtodelim_kafka (
  Timestamp_Ns DateTime64(9,'UTC') CODEC(DoubleDelta, LZ4),
  Hostname LowCardinality(String) CODEC(LZ4),
  Pop LowCardinality(String) CODEC(LZ4),
  Label LowCardinality(String) CODEC(LZ4),
  Tag LowCardinality(String) CODEC(LZ4),
  Poll_Counter UInt64 CODEC(DoubleDelta, LZ4),
  Record_Counter UInt64 CODEC(DoubleDelta, LZ4),
  Function LowCardinality(String) CODEC(LZ4),
  Variable LowCardinality(String) CODEC(LZ4),
  Type LowCardinality(String) CODEC(LZ4),
  Value Float64,
)
ENGINE = Kafka SETTINGS
  kafka_broker_list = 'redpanda-0:9092',
  kafka_topic_list = 'ProtobufListProtodelim',
  kafka_schema = 'prometheus_protolist.proto:PromRecordCounter',
  kafka_max_rows_per_message = 10000,
  kafka_num_consumers = 1,
  kafka_thread_per_consumer = 0,
```

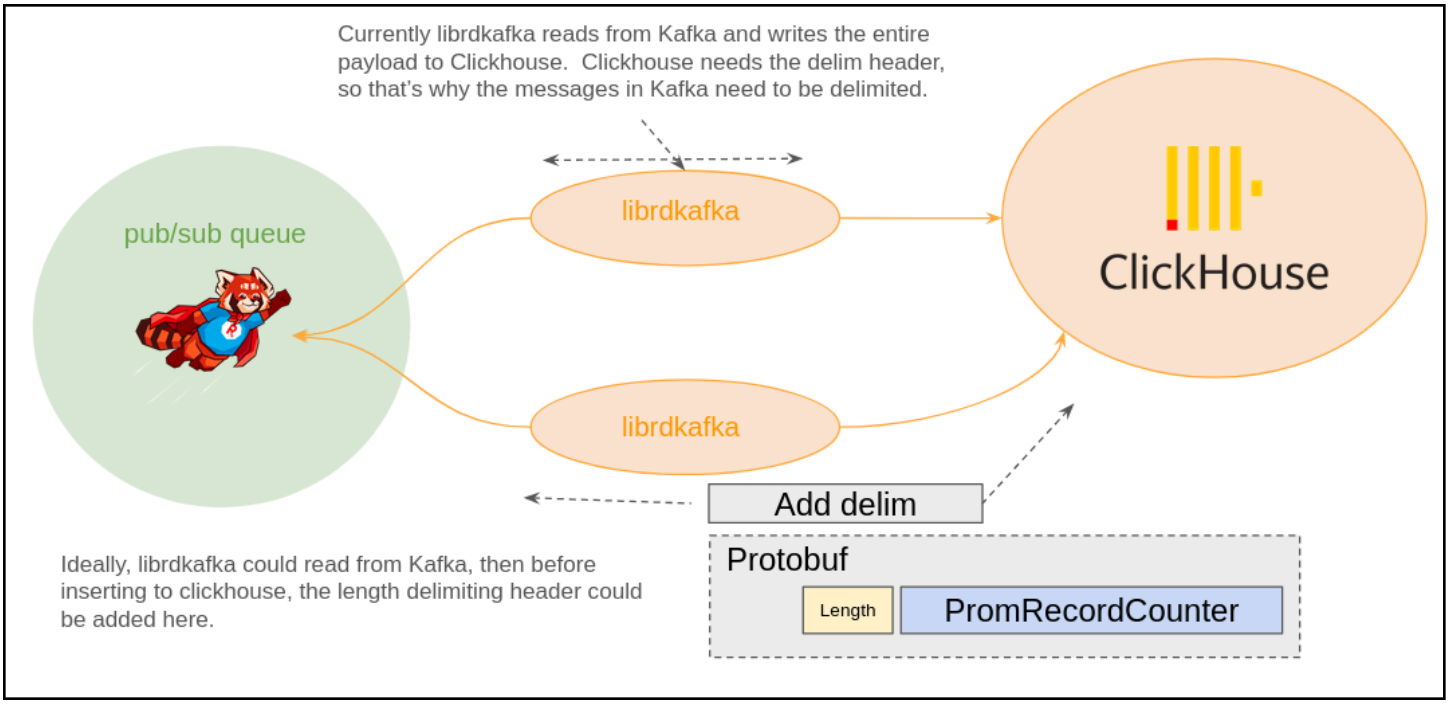
```
  kafka_group_name = 'ProtobufListProtodelim',
  kafka_skip_broken_messages = 1,
  kafka_handle_error_mode = 'stream',
  kafka_format = 'ProtobufList';
```

The kafka broker will need to be changed.
https://github.com/randomizedcoder/gdp/blob/main/build/containers/clickhouse/initdb.d/sql/gdp/ProtobufListProtodelim_kafka.sql#L20

Making data pipelines better

Having spent time thinking about this data pipeline, I've got some ideas how the situation could be improved. Happy to discuss.





Clickhouse Tests

Clickhouse has some really great tests in their repo, which are excellent to refer to as a form of documentation.

What these tests are NOT testing is the Kafka component. I suspect when the inserts are happening in these tests, the clickhouse-client is doing the serialization into the Clickhouse binary protocol, and that's why it works. If anyone from clickhouse can confirm this theory, that would be great.
https://github.com/ClickHouse/ClickHouse/blob/master/tests/queries/0_stateless/02240_protobuflist_format_persons.sh
https://github.com/ClickHouse/ClickHouse/blob/master/tests/queries/0_stateless/format_schemas/02240_protobuflist_format_persons_syntax2.proto

Skipping ProtobufList
https://github.com/ClickHouse/ClickHouse/blob/85218b56f26754828c8d011ebaf782b44633[…]s/integration/test_storage_kafka/test_produce_http_interface.py
https://github.com/ClickHouse/ClickHouse/blob/master/tests/integration/test_storage_kafka/test_produce_http_interface.py#L172

Clickhouse Bulk Inserts

https://clickhouse.com/docs/optimize/bulk-inserts

Clickhouse Kafka Debugging

There isn't a lot of information online about how to debug Clickhouse kafka.

I've found a few things that can help. To make these command easy to find, I appended them in the comments to each of the _kafka.sql files generated

```
-- SHOW CREATE TABLE gdp.ProtobufList_kafka;
-- SELECT * FROM system.kafka_consumers FORMAT Vertical;
-- DETACH TABLE gdp.ProtobufList_kafka;
-- SELECT * FROM gdp.ProtobufList_kafka LIMIT 20;
```

https://github.com/randomizedcoder/gdp/blob/main/build/containers/clickhouse/initdb.d/sql/gdp/ProtobufListKafka_kafka.sql#L31

Clickhouse Kafka Debugging - Verify table create

Once the sql has been run to create the Kafka table, you can verify that it looks good by doing:

```
SHOW CREATE TABLE gdp.ProtobufList_kafka;
```

Clickhouse Kafka Debugging - Kafka Consumers

One really helpful command to see what's going on with Kafka is to do this command. It will show you errors, and various stats.

```
SELECT * FROM system.kafka_consumers FORMAT Vertical;
```

Command to view the detailed Kafka stats from librdkafka

The other thing to check is the clickhouse server error log.

```
docker exec -ti gdp-clickhouse-1 tail -n 30 -f /var/log/clickhouse-server/clickhouse-server.err.log
```

Check clickhouse error log

When Kafka is actually working, you can see it getting busy in the non-error log, because I've got the full kafka logging enabled.

```
docker exec -ti gdp-clickhouse-1 tail -n 30 -f /var/log/clickhouse-server/clickhouse-server.log
```

Check clickhouse normal log

Clickhouse Kafka Debugging - Kafka Events Counters

Another really interesting way to see what's going on is to look at the Kafka event counters in the system database.

Clickhouse does expose some Kafka related counters in the system.events table.

```
477f3e94967e :) SELECT
       event,
       value,
       description
FROM system.events
WHERE event LIKE '%Kafka%';

SELECT
       event,
       value,
       description
FROM system.events
WHERE event LIKE '%Kafka%'

Query id: 8a8a547d-a541-4a82-8645-11e18bf20311


┌─event──────────────────────┬─value─┬─description──────────────────────────────────────────────────────────────────────────────────────┐
1. │ KafkaRebalanceAssignments │     1 │ Number of partition assignments (the final stage of consumer group rebalance)                │
2. │ KafkaMessagesPolled       │    17 │ Number of Kafka messages polled from librdkafka to ClickHouse                                │
3. │ KafkaMessagesRead         │     6 │ Number of Kafka messages already processed by ClickHouse                                     │
4. │ KafkaMessagesFailed       │     5 │ Number of Kafka messages ClickHouse failed to parse                                          │
5. │ KafkaRowsRead             │     5 │ Number of rows parsed from Kafka messages                                                    │
6. │ KafkaBackgroundReads      │     1 │ Number of background reads populating materialized views from Kafka since server start │
└────────────────────────────┴───────┴──────────────────────────────────────────────────────────────────────────────────────────────┘

6 rows in set. Elapsed: 0.003 sec.
```

Example output showing the Clickhouse Kafka events

This SQL is in the file:
https://github.com/randomizedcoder/gdp/blob/main/build/containers/clickhouse/initdb.d/sql/select_system_events_like_kafka.sql

Miscellaneous

No good document is complete without a miscellaneous section!

Here are a few random notes in no particular order:

- CLI flags
  - Gdp has a lot of cli flags:
    - https://github.com/randomizedcoder/gdp/blob/main/cmd/gdp/gdp.go#L83
- Environment variables
  - Most of the cli flags can be overridden by environment variables
  - So essentially, you can override in the docker-compose.yml
    - https://github.com/randomizedcoder/gdp/blob/main/cmd/gdp/gdp.go#L375
- Profiling
  - Profiling can be enabled easily via cli options
    - https://github.com/randomizedcoder/gdp/blob/main/cmd/gdp/gdp.go#L209
  - See also (thanks Dave Cheney!)
    - github.com/pkg/profile
    - https://dave.cheney.net/2013/07/07/introducing-profile-super-simple-profiling-for-go-programs

- Reading protos from Kafka
    - There is a small example program to read the messages from Kafka and print them as JSONs
    - Read the makefile
        - https://github.com/randomizedcoder/gdp/blob/main/cmd/gdp_kafka_client/gdp_kafka_client.go
- Example of franz-go serdes stuff
    - Here's another (even though old & unmaintained) example:
        - https://github.com/cloudhut/owl-shop/blob/7095131ece7a0fee9a58d00b4fbc9f820a0d13be/pkg/shop/order_service.go#L179-L191
        - https://github.com/cloudhut/owl-shop/blob/7095131ece7a0fee9a58d00b4fbc9f820a0d13be/pkg/shop/order_service.go#L236-L307
        - https://github.com/cloudhut/owl-shop/blob/7095131ece7a0fee9a58d00b4fbc9f820a0d13be/pkg/shop/order_service.go#L478-L508
- Redpanda schema decode code here
    - https://github.com/redpanda-data/redpanda/blob/93edacb1d4c802c47d239cf7bbdc1660c869bd01/src/v/datalake/record_schema_resolver.cc#L304
    - https://github.com/redpanda-data/redpanda/blob/dev/src/v/datalake/schema_registry.cc#L20
- Delve debugging
    - In a pit of despair, I went looking to work out what was going wrong, so I added delve capabilities
        - https://github.com/go-delve/delve
    - Turns out that because in Go you often have multiple goroutines doing things concurrently, breakpoints are less useful than you might hope for. Maybe there's some way to say stop only this thread, I don't know.
    - Anyway, hopefully if anyone want to see how to use Delve with a container they can copy this
        - "make build_and_deploy_delve" should deploy the delve version, and then you need to connect Vscode to delve for gdp to start
            - https://github.com/randomizedcoder/gdp/blob/main/Makefile#L28
        - Adding delve to a container here:
            - https://github.com/randomizedcoder/gdp/blob/main/build/containers/gdp/Containerfile.delve#L35
        - Docker-compose tweaks
            - https://github.com/randomizedcoder/gdp/blob/main/build/containers/docker-compose-delve.yml#L208
            - https://github.com/randomizedcoder/gdp/blob/main/build/containers/docker-compose-delve.yml#L225
- Clickhouse StorageKafka Warning
    - There's a funny warning in the error log, like this
        - `"2025.04.04 17:12:52.448355 [ 1396 ] {} <Warning> StorageKafka (ProtobufListKafkaProtodelim_kafka): sasl.kerberos.kinit.cmd configuration parameter is ignored."`
    - I looked at the code at some point, and I think it *always* says this, and it can be safely ignored
- Kafka library
    - https://cwiki.apache.org/confluence/display/KAFKA/KIP-82+-+Add+Record+Headers
    - https://github.com/confluentinc/librdkafka/blob/8e83ba256dd2596c3870ec8b310beaf9c890938f/INTRODUCTION.md?plain=1#L1969
    - Schema registry
        - The libserdes does appear to support writing the kafka header via serdes_framing_cp1_write
        - https://github.com/confluentinc/libserdes/blob/8cf97f7395bf5131d14bacfe896c6a5731b1f0c8/src/framing.c#L38