streamdal | Data Council

Real-Time Schema Discovery

Austin, TX | March, 2023

Daniel Selans
Co-Founder & CTO / **Streamdal**

Daniel Selans

CO-FOUNDER AND CTO / streamdal

- 20+ years of software and systems experience
- Past 10 years spent writing backend Go
- All about event driven, event sourcing and all things async
- Reside in Portland, OR (originally from Riga, Latvia 🇱🇻)

PREVIOUSLY AT

New Relic · Community · DigitalOcean · invision

Why real-time schema discovery?

Why real-time schema discovery?

Need to start with some context

What is a schema?

What is a schema?

Pre-defined and agreed-upon structure and layout for data

Why is it important?

Why is it important?

Data used by many teams;
everyone needs data

Why is it important?

Data used by many teams; everyone needs data

Data Engineers → AI, machine learning, analytics

Why is it important?

Data used by many teams; everyone needs data

Data Engineers → AI, machine learning, analytics

Developers → Build features

Why is it important?

Data used by many teams; everyone needs data

Data Engineers → AI, machine learning, analytics

Developers → Build features

Platform & DevOps → Troubleshooting/debug

What happens without schema?

What happens without schema?

Incidents & outages

Unexpected upstream schema changes impact downstream consumers

What happens without schema?

Incidents & outages

Unexpected upstream schema changes impact downstream consumers

Duplicate effort

Everyone normalizes data according to their requirements

What happens without schema?

Incidents & outages

Unexpected upstream schema changes impact downstream consumers

Duplicate effort

Everyone normalizes data according to their requirements

Wasted time

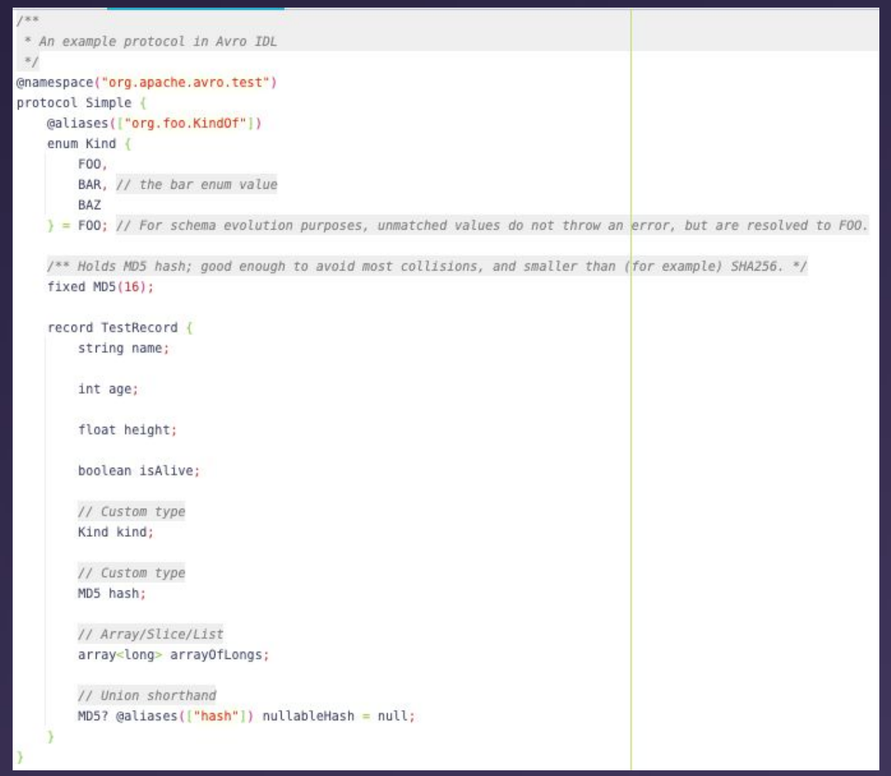
Tool and process updates

Schema examples

cloudevents

```
{
  "data":{ EVENT_DATA },
  "datacontenttype": "application/json",
  "id": "MESSAGE_ID",
  "source": "//storage.googleapis.com/projects/_/buckets/BUCKET_NAME",
  "specversion": "1.0",
  "type": "google.cloud.storage.object.v1.finalized",
  "time": "EVENT_GENERATION_TIME",
  "subject": "objects/my-file.txt",
}
```

Schema examples



```
/**
 * An example protocol in Avro IDL
 */
@namespace("org.apache.avro.test")
protocol Simple {
    @aliases(["org.foo.KindOf"])
    enum Kind {
        FOO,
        BAR, // the bar enum value
        BAZ
    } = FOO; // For schema evolution purposes, unmatched values do not throw an error, but are resolved to FOO.

    /** Holds MD5 hash; good enough to avoid most collisions, and smaller than (for example) SHA256. */
    fixed MD5(16);

    record TestRecord {
        string name;

        int age;

        float height;

        boolean isAlive;

        // Custom type
        Kind kind;

        // Custom type
        MD5 hash;

        // Array/Slice/List
        array<long> arrayOfLongs;

        // Union shorthand
        MD5? @aliases(["hash"]) nullableHash = null;
    }
}
```

Schema examples

protobuf
Protocol Buffers

43 lines (37 sloc) | 857 Bytes

```proto
syntax = "proto3";

package fakes;

import "billing.proto";
import "billingv2.proto";
import "coin.proto";
import "post.proto";
import "product.proto";
import "search.proto";
import "user.proto";
import "weather.proto";

option go_package = "github.com/batchcorp/schemas/build/go/events/fakes";

enum EventType {
  EVENT_TYPE_UNSET = 0;
  EVENT_TYPE_BILLING = 1;
  EVENT_TYPE_SEARCH = 4;
  EVENT_TYPE_PRODUCTS = 10;
  EVENT_TYPE_USERS = 11;
  EVENT_TYPE_POSTS = 12;
  EVENT_TYPE_COINS = 13;
  EVENT_TYPE_WEATHER = 14;
}

message Event {
  EventType type = 1;
  int64 timestamp_nano = 2;
  string request_id = 3;
  string source = 4;

  oneof event {
    Billing billing = 100;
    Search search = 101;
    Product product = 102;
    User user = 103;
    Post post = 104;
    Coin coin = 105;
    Weather weather = 106;
    BillingV2 billing_v2 = 107;
  }
}
```

21 lines (17 sloc) | 409 Bytes

```proto
syntax = "proto3";

package fakes;

option go_package = "github.com/batchcorp/schemas/build/go/events/fakes";

// Used for fake event generation
message Billing {
  string order_id = 1;
  string order_state = 2;
  string customer_id = 3;
  int64 order_date = 4;
  repeated BillingProduct products = 5;
}

message BillingProduct {
  string id = 1;
  string name = 2;
  int32 quantity = 3;
  float price = 4;
}
```

Conclusion

- Having a schema is a foundational component of your platform
- Lack of a schema will result in:
  - Waste of engineering resources
  - Slower feature & work delivery
  - Incidents & Outages
    - Difficult to debug
    - Slower mean time to recovery

WANT: Solid schema process

STEP 1

Define schema

WANT: Solid schema process

STEP 1

Define schema

...but who's the source of truth?

WANT: Solid schema process

STEP 1

Define schema

...but who's the source of truth?

STEP 2

Establish schema update process

WANT: Solid schema process

STEP 1

Define schema

...but who's the source of truth?

STEP 2

Establish schema update process

...but what if someone doesn't follow it?

WANT: Solid schema process

STEP 1

Define schema

...but who's the source of truth?

STEP 2

Establish schema update process

...but what if someone doesn't follow it?

STEP 3

Real-time Schema Discovery

Real-Time Schema Discovery

In 17 easy steps 🥲

DIY Real-Time Schema Discovery

DIY Real-Time Schema Discovery

- Determine and nominate one or more sources of truth for data (ie. Kafka)

DIY Real-Time Schema Discovery

- Determine and nominate one or more sources of truth for data (ie. Kafka)
- Write a service that consumes from Kafka

DIY Real-Time Schema Discovery

- Determine and nominate one or more sources of truth for data (ie. Kafka)
- Write a service that consumes from Kafka
- Write a library that parses JSON and generates a typed schema (ie. parquet)

DIY Real-Time Schema Discovery

- Determine and nominate one or more sources of truth for data (ie. Kafka)
- Write a service that consumes from Kafka
- Write a library that parses JSON and generates a typed schema (ie. parquet)
- Research and implement schema conflict resolution

DIY Real-Time Schema Discovery

- Determine and nominate one or more sources of truth for data (ie. Kafka)
- Write a service that consumes from Kafka
- Write a library that parses JSON and generates a typed schema (ie. parquet)
- Research and implement schema conflict resolution
- Implement distributed schema election support in service (learn raft)

DIY Real-Time Schema Discovery

- Determine and nominate one or more sources of truth for data (ie. Kafka)
- Write a service that consumes from Kafka
- Write a library that parses JSON and generates a typed schema (ie. parquet)
- Research and implement schema conflict resolution
- Implement distributed schema election support in service (learn raft)
- Write a distributed schema-election service

DIY Real-Time Schema Discovery

- Determine and nominate one or more sources of truth for data (ie. Kafka)
- Write a service that consumes from Kafka
- Write a library that parses JSON and generates a typed schema (ie. parquet)
- Research and implement schema conflict resolution
- Implement distributed schema election support in service (learn raft)
- Write a distributed schema-election service
- Figure out what to do during unresolvable schema conflicts (dead-letter?)

DIY Real-Time Schema Discovery

- Determine and nominate one or more sources of truth for data (ie. Kafka)
- Write a service that consumes from Kafka
- Write a library that parses JSON and generates a typed schema (ie. parquet)
- Research and implement schema conflict resolution
- Implement distributed schema election support in service (learn raft)
- Write a distributed schema-election service
- Figure out what to do during unresolvable schema conflicts (dead-letter?)
- And while you're at it, probably patent something you discovered while building this 😘

We actually did all of this

We actually did all of this

and so did other folks... 😓

Companies with schema inference support

BigQuery

And several others ...

Implementation cost

Implementation cost

- 6 months to build solution

Implementation cost

- 6 months to build solution
- 4 months to get it “ready for prod”

Implementation cost

- 6 months to build solution
- 4 months to get it “ready for prod”
- 3 months to deal with edge cases

Implementation cost

- 6 months to build solution
- 4 months to get it “ready for prod”
- 3 months to deal with edge cases
- 1 week every 2 months for troubleshooting/debug of “strange” behavior

Implementation cost

- 6 months to build solution
- 4 months to get it “ready for prod”
- 3 months to deal with edge cases
- 1 week every 2 months for troubleshooting/debug of “strange” behavior

- 2 principal-level engineers for 1.2 years ($200K+/year)

Implementation cost

- 6 months to build solution
- 4 months to get it “ready for prod”
- 3 months to deal with edge cases
- 1 week every 2 months for troubleshooting/debug of “strange” behavior

- 2 principal-level engineers for 1.2 years ($200K+/year)
- 2 senior-level engineers for 8 hrs per month – ongoing support (~$150K/year)

Implementation cost

- 6 months to build solution
- 4 months to get it “ready for prod”
- 3 months to deal with edge cases
- 1 week every 2 months for troubleshooting/debug of “strange” behavior

- 2 principal-level engineers for 1.2 years ($200K+/year)
- 2 senior-level engineers for 8 hrs per month – ongoing support (~$150K/year)
- ~$500K first year spend; ~$40K/year ongoing spend

Or in other words...

Don’t do it!

(probably)

Instead, establish a **Foundational Schema**

Step 1: Intro to protobuf

> Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – think XML, but smaller, faster, and simpler.

PROTO DEFINITION

```
syntax = "proto3";

package flags;

message SubCommand {
  string action = 1;
  repeated string args = 2;
  int64 count = 3;
}

message Base {
  string action = 1;
  SubCommand sub = 2;
  bool debug = 3;
  string version = 4;
}
```

Some other serialization formats you may have heard of: Thrift, Flatbuffer, Avro.

Step 2: Intro to protobuf

COMPILE SCHEMAS

- You compile the .protos with `protoc` tool (part of protobuf toolchain)
- `protoc` generates “compiled” *.pb.go files which contain generated code for accessing the attributes and structures you defined in .protos

```
docker run --rm -w $(PWD) -v $(PWD):$(PWD) -w${PWD} jaegertracing/protobuf:0.2.0 \
--proto_path=./protos/args \
--go_out=plugins=grpc:$(GO_PROTOS_DIR)/args \
--go_opt=paths=source_relative \
-o ./$(GO_DESCRIPTOR_SET_DIR)/args.fds \
--include_imports \
--include_source_info \
protos/args/*.proto
```

SCREENSHOT OF COMPILING PROTOS IN DOCKER

Step 3: Intro to protobuf

What does the generated protobuf look like?

PROTO DEFINITION

```
syntax = "proto3";

package flags;

message SubCommand {
  string action = 1;
  repeated string args = 2;
  int64 count = 3;
}

message Base {
  string action = 1;
  SubCommand sub = 2;
  bool debug = 3;
  string version = 4;
}
```

COMPILED

```
type SubCommand struct {
    Action string
    Args   []string
    Count  int64
}

type Base struct {
    Action  string
    Sub     SubCommand
    Debug   bool
    Version string
}
```

Schema Management Flow

STEP 1

Protobuf schema lives in a Github repo

STEP 2

Repo is governed, protected by architects & stakeholders

STEP 3

Schema changes are proposed via PR

STEP 4

Once merged, Github kicks off notification about schema update

STEP 5

Stakeholders see schema update notification and update their components

(with updated schema)

STEP 6

Backwards incompatible changes are NOT permitted

(During PR process)

STEP 7

Consumers are deployed with new schema

STEP 8

Producers are deployed with new schema

What does it look like in practice?

STEP 1

All services use common schema

STEP 2

Services encode ingres data as protobuf

STEP 3

Services emit encoded data to data bus

(kafka, pulsar, nats, etc)

STEP 4

Consumers read data from data bus and know how to decode data due to schema

STEP 5

Consumers do XYZ with decoded data

1

Single source of truth

Everyone knows where to find out what data is supposed to look like.

1

Single source of truth

Everyone knows where to find out what data is supposed to look like.

2

No guesswork surrounding data structure

Team A does not need to find Person X on Team Y who knows about possible field values in a certain data structure.

1

Single source of truth

Everyone knows where to find out what data is supposed to look like.

2

No guesswork surrounding data structure

Team A does not need to find Person X on Team Y who knows about possible field values in a certain data structure.

3

No need for normalization and/or sanitization step

Everyone knows where to find out what data is supposed to look like.

1

Single source of truth

Everyone knows where to find out what data is supposed to look like.

2

No guesswork surrounding data structure

Team A does not need to find Person X on Team Y who knows about possible field values in a certain data structure.

3

No need for normalization and/or sanitization step

Everyone knows where to find out what data is supposed to look like.

4

Cost reduction

Less data to transport (ie. via Kafka) –
75% size reduction VS JSON
Smaller memory footprint
Lower network costs

1

Single source of truth

Everyone knows where to find out what data is supposed to look like.

2

No guesswork surrounding data structure

Team A does not need to find Person X on Team Y who knows about possible field values in a certain data structure.

3

No need for normalization and/or sanitization step

Everyone knows where to find out what data is supposed to look like.

4

Cost reduction

Less data to transport (ie. via Kafka) – 75% size reduction VS JSON
Smaller memory footprint
Lower network costs

5

Less Bugs

Reduce number of data structure related mistakes

1

Single source of truth

Everyone knows where to find out what data is supposed to look like.

2

No guesswork surrounding data structure

Team A does not need to find Person X on Team Y who knows about possible field values in a certain data structure.

3

No need for normalization and/or sanitization step

Everyone knows where to find out what data is supposed to look like.

4

Cost reduction

Less data to transport (ie. via Kafka) –
75% size reduction VS JSON
Smaller memory footprint
Lower network costs

5

Less Bugs

Reduce number of data structure related mistakes

6

MORE IMPORTANTLY

Establish a solid foundation for the future

Data structure affects all of engineering. Adopting a solid schema process will continue to pay huge dividends in every part of your org.

Conclusion (1)

- DIY schema discovery in distributed systems is **really** hard

Conclusion (1)

- DIY schema discovery in distributed systems is **really** hard
    a. Hardest part == conflict resolution

Conclusion (1)

- DIY schema discovery in distributed systems is **really** hard
    a. Hardest part == conflict resolution
    b. Huge time sink

Conclusion (1)

- DIY schema discovery in distributed systems is **really** hard
    a. Hardest part == conflict resolution
    b. Huge time sink
    c. Only necessary if you use a semi-structured data format (JSON)

Conclusion (1)

- DIY schema discovery in distributed systems is **really** hard
    a. Hardest part == conflict resolution
    b. Huge time sink
    c. Only necessary if you use a semi-structured data format (JSON)
- Time better spent adopting a strong schema format (Protobuf)

Conclusion (1)

- DIY schema discovery in distributed systems is **really** hard
    a. Hardest part == conflict resolution
    b. Huge time sink
    c. Only necessary if you use a semi-structured data format (JSON)
- Time better spent adopting a strong schema format (Protobuf)
    a. But... will require organization-wide adoption

Conclusion (2)

- You should spend time on DIY schema discovery IF:

Conclusion (2)

- You should spend time on DIY schema discovery IF:
    a. You have **NO** control over ingres data structure

Conclusion (2)

- You should spend time on DIY schema discovery IF:
    a. You have **NO** control over ingres data structure
    b. ... And even then, the ingres data should be wrapped in protobuf

Conclusion (2)

- You should spend time on DIY schema discovery IF:
    a. You have **NO** control over ingres data structure
    b. ... And even then, the ingres data should be wrapped in protobuf
- Adopting protobuf is easier at small orgs

Conclusion (2)

- You should spend time on DIY schema discovery IF:
    - a. You have **NO** control over ingres data structure
    - b. ... And even then, the ingres data should be wrapped in protobuf
- Adopting protobuf is easier at small orgs
    - a. Large orgs need to do piecemeal adoption (which is harder)

Thank you!

Daniel Selans

Co-Founder & CTO / **Streamdal**

STREAMDAL.COM

Let's stay in touch!

daniel@streamdal.com

linkedin.com/in/dselans/

github.com/dselans

github.com/batchcorp/plumber