Event Data Platform

or, JSON is okay too
Andrew Otto
Principal Software Engineer
at The Wikimedia Foundation
(previously an SRE)
The Plan
What makes Wikimedia unique?
What makes Wikimedia unique?

A typical event streaming stack.
The Plan

What makes Wikimedia unique?

A typical event streaming stack.

Why this stack doesn’t work for WMF.
The Plan

What makes Wikimedia unique?

A typical event streaming stack.

Why this stack doesn’t work for WMF.

WMF’s solution.
not just Wikipedia
What makes WIKIMEDIA unique?
Free and open source
Data should be public!

Privacy should be protected
Distributed
Conflict!
A typical Kafka event streaming stack
event bus
Producers & Consumers

client libraries
Producers & Consumers

Stream Processing
(Kafka Streams, Flink, etc.)

stream processing framework
Kafka

REST Proxy

Producers & Consumers

Kafka Connect

Connectors

Stream Processing
(Kafka Streams, Flink, etc.)

ingestion framework
Kafka

REST Proxy

Schema Registry

Avro

Producers & Consumers

Stream Processing

(Kafka Streams, Flink, etc.)

Kafka Connect

Connectors
Free and open source

“As an organization, we strive to use open source tools over proprietary ones”

The first rule of open source

1. Free Redistribution

The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.
Confluent Community License
“[...] does not allow you to provide the software as a SaaS offering [...]”
Public data

“data we offer for re-use MUST use clearly specified data schemas and SHOULD be based on widely used open standards”

https://www.mediawiki.org/wiki/Wikimedia_Engineering_Architecture_Principles#api/schema
Distributed development

“Our software architecture and development processes SHOULD be geared towards building an ecosystem of third-party re-users and all kinds of contributors to the code base [...], volunteers and professionals alike”

Kafka
REST Proxy

Connectors

Producers & Consumers

Schema Registry
Avro

Stream Processing
(Kafka Streams, Flink, etc.)

Kafka Connect

Connectors
To use Avro you need:

- An Avro implementation in your language
To use Avro you need:

- An Avro implementation in your language
- An Avro schema
Avro at rest

Binary data at rest in a file:

schema is in the file

source: https://www.clairvoyant.ai/blog/big-data-file-formats
Avro in streams

In streams, each message is binary data.
In streams, each message is binary data.

To read, you need the Avro schema.
In streams, each message is binary data.

To read, you need the Avro schema.

But, how to get it?
Avro in streams

Confluent’s solution

1. Schemas registered in a **centralized** schema registry and assigned unique integer **schema ID**.
Avro in streams  Confluent’s solution

1. Schemas registered in a **centralized** schema registry and assigned unique integer **schema ID**.

2. Confluent’s serializer: prepends the binary schema ID.
1. Schemas registered in a **centralized** schema registry and assigned unique integer **schema ID**.

2. Confluent’s serializer: prepends the binary schema ID.

3. Confluent’s deserializer: pops the schema ID and gets schema from registry.

1. Schemas registered in a **centralized** schema registry and assigned unique integer **schema ID**.

2. Confluent’s serializer: prepends the binary schema ID.

3. Confluent’s deserializer: uses the schema ID to get schema from registry.

That’s a lot of coupling!
Schema Registry & Avro in streams

Wikimedia’s software development model is distributed
Making choices

Constraints
- Open Source
- Public Data
- Distributed
Alternatives?

If not Avro, then what?
What about good ol’ JSON & JSONSchema?
JSON and Event Schemas
What are schemas for?
What are schemas for?

Schemas are the data contract.

Like an API spec, but for data.
What are schemas for?

Schemas are needed for

1. Data validation
2. Data integration
Schemas are for: Validation

Producing - need to ensure data is valid

E.g. `user_id` is required and must be an integer > 0

```json
{
"user_id": 123,
"create_dt": "2022-10-05T12:00:00Z"
}
```
Automating ingestion - need to know data types

{  
  "user_id": 123,  
  "create_dt": "2022-10-05T12:00:00Z"  
}

CREATE TABLE user (  
  user_id integer,  
  create_dt timestamp  
)
Can we use JSONSchema?

1. Validation
2. Integration
JSON is (arguably) more ubiquitous than Avro
JSON serialization

No special tooling to read JSON serialized data
JSON serialization

JSON is only ‘loosely’ schemaed

```json
{
    "name": "Bobby Tables",
    "age": 12,
    "height": 1.7,
    "create_dt": "2022-10-05T12:00:00Z"
}
```
JSONSchema

```json
{
  "type": "object",
  "properties": {
    "name": {
      "type": "string"
    },
    "age": {
      "type": "integer"
    },
    "height": {
      "type": "number"
    },
    "create_dt": {
      "type": "string",
      "format": "date-time"
    }
  }
}
```

Used to validate JSON data

```json
{
  "name": "Bobby Tables",
  "age": 12,
  "height": 1.7,
  "create_dt": "2022-10-05T12:00:00Z"
}
```
Can be used for automated integration

```json
{  "type": "object",
  "properties": {  
    "name": {  
      "type": "string"
    },  
    "age": {  
      "type": "integer"
    },  
    "height": {  
      "type": "number"
    },  
    "create_dt": {  
      "type": "string",
      "format": "date-time"
    }
  }
}
```

```sql
CREATE TABLE user (  
  name varchar,
  age integer,
  height float,
  create_dt timestamp
)
```
Can we use JSONSchema?

1. Validation  
2. Integration
Can we use JSONSchema?

Avro and Confluent Schema Registry do do fancy stuff
Schema distribution
Confluent Schema Registry + Avro features

Schema distribution

Schema versioning
Confluent Schema Registry + Avro features

- Schema distribution
- Schema versioning
- Schema evolution
JSONSchema distribution
Schema URIs

JSONSchemas have ‘meta schemas’
Schema URIs

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    ...   
  }
}
```
Schema URIs

{(This is a data record, not a schema)
full qualified URL in $schema

```json
{
  "$schema": "http://schema.company.org/entity/user.json",
  "name": "Bobby Tables",
  "age": 12,
  "height": 1.7
}
```
fully qualified URL in $schema

{  
  "$schema": "http://schema.company.org/entity/user.json",
  "name": "Bobby Tables",
  "age": 12,
  "height": 1.7
}

This is not decentralized
Schema Repositories

Repository > Registry
Schema Repositories

Git is great at decentralization.
Schemas in git
Only path part of URI in $schema

```
{
  "$schema": "/entity/user.json",
  "name": "Bobby Tables",
  "age": 12,
  "height": 1.7
}
```
This isn’t enough to locate a schema

{  
  "$schema": "/entity/user.json",
  "name": "Bobby Tables",
  "age": 12,
  "height": 1.7
}
Prefix \texttt{\$schema} with a base address

\texttt{"\$schema": "/entity/user.json"}

\texttt{"file:///local/path/to/schema_repository"} \texttt{+ \$schema}

\texttt{"https://schema.example.org/repository"} \texttt{+ \$schema}

\texttt{"hdfs://namenode-addr/path/to/repository"} \texttt{+ \$schema}
JSONSchema versioning
Versioned Schemas

Code changes
Versioned Schemas

Code changes

so does data
Versioned Schemas

Don’t use git for versioning of schemas
Versioned Schemas

/entity/user/1.1.0.json
**title** keyword is used to associate different schema versions.

```json
/entity/user/1.1.0.json
{
  "title": "entity/user",
  "type": "object",
  "properties": { ... }
}
```
A schema’s unique identity is declared using `$id` keyword

```
/entity/user/1.1.0.json
{
  "$id": "/entity/user/1.1.0",
  "title": "entity/user",
  "type": "object",
  "properties": { ... }
}
```
Versioned Schemas

Event records set
$schema to match their schema’s $id

/path/to/entity/user/1.1.0.json
{
   "$id": "path/to/entity/user/1.1.0",
   "title": "entity/user",
   "type": "object",
   "properties": { ...

   "$schema": "path/to/entity/user/1.1.0",
   "name": "Bobby Tables",
   "age": 12,
   "height": 1.7
}
JSONSchema evolution
COMPATIBILITY + EVOLUTION

ORIGINAL
PRODUCER

NO CHANGE

FORWARDS

REVISED
PRODUCER

REVISED
CONSUMER

BACKWARDS

INCOMPATIBLE

REVOLUTION!

ORIGINAL
CONSUMER

source: Paul Downey - https://www.flickr.com/photos/psd/3352842349
Only allowed to add new optional fields
• No field type changes
JSONSchema compatibility rule

- No field type changes
- No field deletions
No field type changes
No field deletions
No field renames
JSONSchema versioning and evolution

/entity/user/1.0.0.json
/entity/user/1.1.0.json
/entity/user/1.1.1.json
/entity/user/1.2.0.json
jsonschema-tools
a single file for developers to edit

/entity/user/current.json
/entity/user/current.json
{
    "$id": "/entity/user/1.1.0",
    "title": "entity/user",
    "type": "object",
    "properties": { ... }
}

/entity/user/1.1.0.json
entity/user

Major Version 1
✓ 1.1.0 must be compatible with 1.0.0
✓ 1.1.1 must be compatible with 1.1.0
✓ 1.2.0 must be compatible with 1.1.1

tests

entity/user
1.0.0.json
✓ must be a valid JSONSchema
✓ must be a secure JSONSchema
✓ properties must be snake_case
✓ has no union types
✓ all required properties must exist
✓ examples must have $schema == schema's $id
✓ examples must validate against schema
✓ should have minimum and maximum values inside the configured bounds for all numeric fields
Summary
Summary

JSONSchema distribution
Summary

JSONSchema distribution

JSONSchema versioning

/entity/user/1.1.1.1.json
Summary

JSONSchema distribution

JSONSchema versioning

/entity/user/1.1.1.1.json

JSONSchema evolution

managed with jsonschema-tools
Summary

JSONSchema for validation
Summary

JSONSchema for validation

JSONSchema for integration
# Summary

<table>
<thead>
<tr>
<th>PROS</th>
<th>CONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSON is easy to use</td>
<td>Inefficient text format (but compresses well)</td>
</tr>
<tr>
<td>JSONSchema is common outside of ‘big data’</td>
<td>Opinionated about event content ($schema)</td>
</tr>
<tr>
<td>schemas are code, not data</td>
<td>Rigid schema evolution rules</td>
</tr>
<tr>
<td>No centralized registry</td>
<td>Lack of big data integration tools (schema converters)</td>
</tr>
<tr>
<td>More decoupled</td>
<td></td>
</tr>
<tr>
<td>Simple schema evolution</td>
<td></td>
</tr>
<tr>
<td>Don’t need schemas to read data</td>
<td></td>
</tr>
</tbody>
</table>
**WMF Event Platform Links**

- [Event Platform blog post series](#)
- [Event Platform Docs](#)
- [schema.wikimedia.org](#) - event schemas service and UI
- [stream.wikimedia.org](#) - public stream data API and UI
Questions?

- Andrew Otto
  otto@wikimedia.org