Bringing Avro and AsyncAPI together: Pitfalls and Learnings

September 22 - 24, 2025

Dr. Annegret Junker &
Fabrizio Lazzaretti

codecentric WAVESTONE

Dr. Annegret Junker

Architect at codecentric

Interested in Microservices, DDD, API Design, Event Design and how to get faster from specification to code

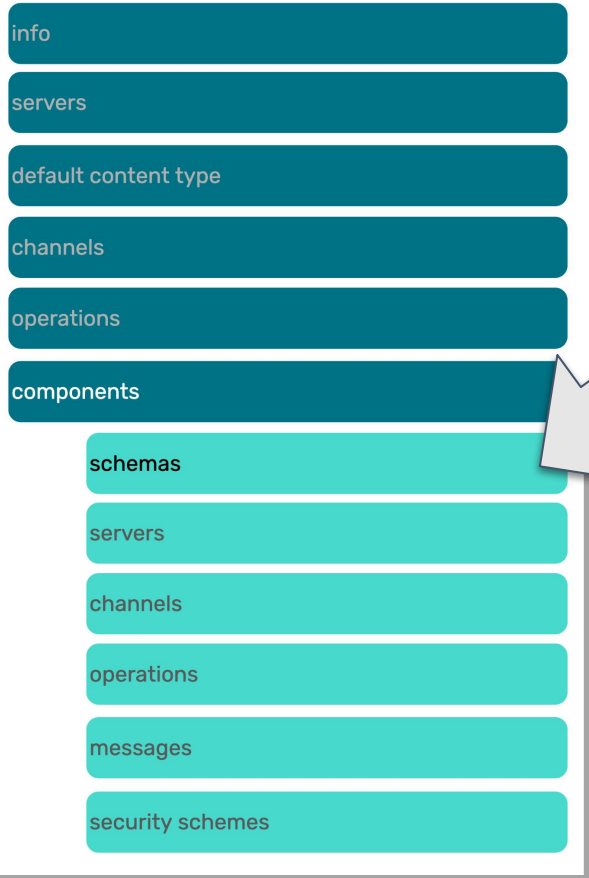
Fabrizio Lazzaretti

Architect at Wavestone

interested in API Design, Event Design, Microservices and how to make a developer's life easier

codecentric WAVESTONE

ASYNC API



- AsyncAPI is the defacto standard to define events in asynchronous communication.
- It allows using different data formats like JSON, XML, and Avro.
- The schemas of those can be defined inside of an AsyncAPI as JSON or YAML.

AsyncAPI Example

```yaml
channels:
  BookPurchasedEventAvroChannel:
    description: Channel where messages are stored when a book was
purchased in Avro
    address: book-purchased-avro
    messages:
      BookPurchasedEventAvro:
        $ref: '#/components/messages/BookPurchasedEventAvro'
    bindings:
      kafka:
        topic: 'book-purchased-avro'
        bindingVersion: '0.5.0'
operations: …
components:
  messages:
    BookPurchasedEventAvro:
      headers:
        $ref: '#/components/schemas/MessageHeader'
      payload:
        schemaFormat: 'application/vnd.apache.avro+yaml;version=1.9.
0'
        schema:
          type: record
          name: Book
          namespace: com.example
          doc: A schema representing a book and its metadata for the
library for, e.g., search results
```

https://github.com/DDAPID/code-samples/blob/main/chapter-6-4/asyncapi-avro.yaml

AsyncAPI part

Avro part

codecentric WAVESTONE

Linking Schemas

Schemas can also be linked **externally**.

- Better integration with editors

However

- Separat versioning that can lead to **problems**, no single source of truth
- Potential multiple **dependencies** need to be handled

```
payload:
  schemaFormat: application/vnd.apache.avro;version=1.9.0
  schema:
    $ref: "https://www.asyncapi.com/resources/casestudies/adeo/CostingRequestPayload.avsc"
```

What is AVRO?

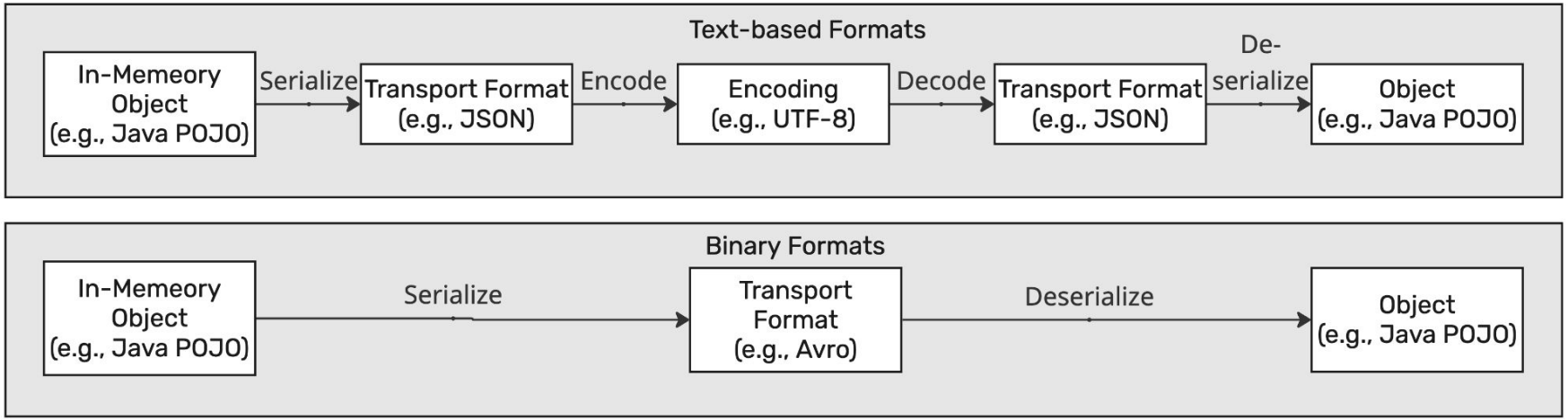
© Galerie der Moderne, Munich

Apache Avro is a binary-serialization format developed in 2009 as a sub-project of Hadoop. Protobuf and Apache Thrift were not considered optimal for the needs of the Hadoop project

M. Kleppmann, Designing data-intensive applications, The big ideas behind reliable, scalable, and maintainable systems, First edition. Beijing: O'Reilley, 2017, 1590 pp.,
D. Cutting, [proposal] new subproject: Avro, Apache Mail Archives, 2009. [Online]. Available: https://lists.apache.org/thread/z571w0r5jmfsjvnl0fq4fgg0vh28d3bk (visited on 09/11/2024).

Currently in version 1.12 (AsyncAPI Studio only supports 1.9)

What is Avro? Binary vs. Text Based Formats



Junker, A.; Lazzaretti, F.: Crafting Great APIs using DDD, apress 2025

What is AVRO?

© George Braque, Clarinet and a Bottle of Rum on a Mantelpiece 1911, Tate Modern

primitive types: null, boolean, int, long, float, double, bytes, string

complex types: record, enum, array, map, fixed

Schemas are defined in **Avro IDL** in JSON.

Avro Has Two Encoding Types

© Pablo Picasso, Woman dressing Her Hair, 1940, Guggenheim New York

"Avro specifies two serialization encodings: binary and JSON."
Most applications: binary encoding, as it is smaller and faster.
But, for debugging and web-based applications, the JSON encoding may sometimes be appropriate."

https://avro.apache.org/docs/1.12.0/specification/#encodings

Avro Example

```
{
  "type": "record",
  "name": "Book",
  "namespace": "com.example",
  "doc": "A schema representing a book and its metadata for the library for,
  "fields": [
    {
      "name": "isbn13",
      "type": "string",
      "doc": "The 13-digit ISBN, e.g., 9781234567890"
    },
    {
      "name": "title",
      "type": "string",
      "doc": "The title of the book"
    },
```

https://github.com/DDAPID/code-samples/blob/main/chapter-6-2/book.avsc

Pitfalls: Examples

```
"authors": {
  "type": "array",
  "items": {
    "type": "object",
    "properties": {
      "name": {
        "type": "string",
        "minLength": 1,
        "maxLength": 100,
        "examples": [ "Eric Evans", "Dr. John Smith"],
        "description": "The full name of the author with titles"
      },
```

JSON Schema

https://github.com/DDAPID/code-samples/blob/main/chapter-6-2/book.schema.json

Pitfalls: Examples

AVRO

```
"fields": [
  {
    "name": "name",
    "type": "string",
    "doc": "The full name of the author with titles, e.g.: \"Eric Evans\",
```

There is no explicit support of examples - they need to be mentioned in the description.

Pitfalls: Optionals

```
"required": [
  "isbn13",
  "title",
  "authors",
  "numberOfTextPosition",
  "category",
  "tags",
  "hasValidLicense",
  "publishingDate"
],
```

JSON Schema

Required fields are defined in JSON directly.

https://github.com/DDAPID/code-samples/blob/main/chapter-6-2/book.schema.json

Pitfalls: Optionals

```
{
  "name": "coverThumbnailWebP",
  "type": ["null", "bytes"],
  "default": null,
  "doc": "Base64 encoded WebP image of the book cover thumbnail"
},
```

You can define optionals.
Null has to be first, it is the default (as it should be for optionals :-))
But that is not checked in AsyncAPI Studio.

Pitfalls: Description and Doc

```yaml
channels:
  BookPurchasedEventAvroChannel:
    description: Channel where messages are stored when a book was
purchased in Avro
    address: book-purchased-avro
    messages:
      BookPurchasedEventAvro:
        $ref: '#/components/messages/BookPurchasedEventAvro'
    bindings:
      kafka:
        topic: 'book-purchased-avro'
        bindingVersion: '0.5.0'
operations: …
components:
  messages:
    BookPurchasedEventAvro:
      headers:
        $ref: '#/components/schemas/MessageHeader'
      payload:
        schemaFormat: 'application/vnd.apache.avro+yaml;version=1.9.
0'
        schema:
          type: record
          name: Book
          namespace: com.example
          doc: A schema representing a book and its metadata for the
library for, e.g., search results
```

https://github.com/DDAPID/code-samples/blob/main/chapter-6-4/asyncapi-avro.yaml

AsyncAPI part

Use `description` in AsyncAPI part and `doc` in Avro schema definition part.

Avro part

codecentric WAVESTONE

How do we now use the schema?

With a JSON formatted schema file (*.avsc), single schema file, we can:

- Publish the schema to a registry (apicurio, Confluent Schema Registry, …)
- Generate Code
- Use Avro compilers

We need to extract the schema out of the AsyncAPI

Currenlty, we use a custom script `extract-avro.sh`.

Do you know better ways?

https://github.com/DDAPID/code-samples/blob/main/chapter-6-2/book.avsc

Schema Evolution

© Refik Anadol: Unsupervised, 2022, MoMA, New York

An essential aspect when starting to use a format is to also think about a schema evolution strategy. The schema evolution describes ***how a schema can change over time and when a breaking change happens.*** *Protobuf and Apache Avro have special mechanisms that allow the* ***modification of property*** *names without breaking consumers' and producers' implementations or* ***runtime behavior****. These aspects are particularly important when implementing event sourcing, where you want to be a****ble to read new events and old events*** *that were produced weeks or even years ago in parallel.*

Schema

M. Kleppmann, Designing data-intensive applications, The big ideas behind reliable, scalable, and maintainable systems, First edition. Beijing: O'Reilley, 2017, 1590 pp.,

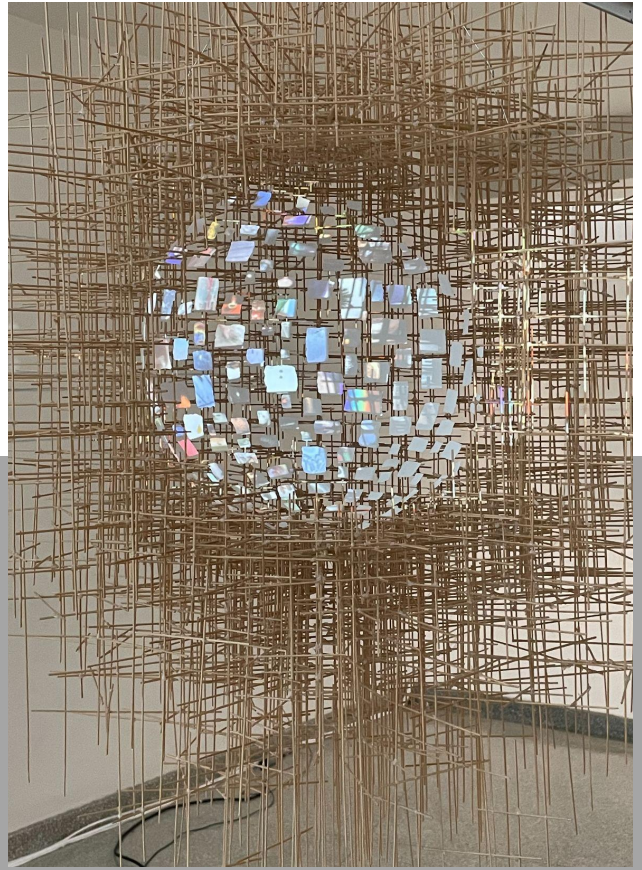
Schema Evolution: Breaking Changes

Semantic changes The semantic description changes the meaning. A semantic change can happen in a comment or description of an API without touching the specification in code.
Syntactic changes A syntactic change is in the specification itself. Such a change can be validated by tools.

The main criteria for breaking and non-breaking changes are the same; however, there are a few more non-breaking changes in Apache Avro. Field names can be changed, and aliases can still be used to be compatible with the old version.
In addition, default values can cover the case when a field is not set. However, the default field does not make the field optional.

Specification | apache avro, 2022. [Online]. Available: https://avro.apache.org/docs/1.11.1/specification/ (visited on 08/24/2024).

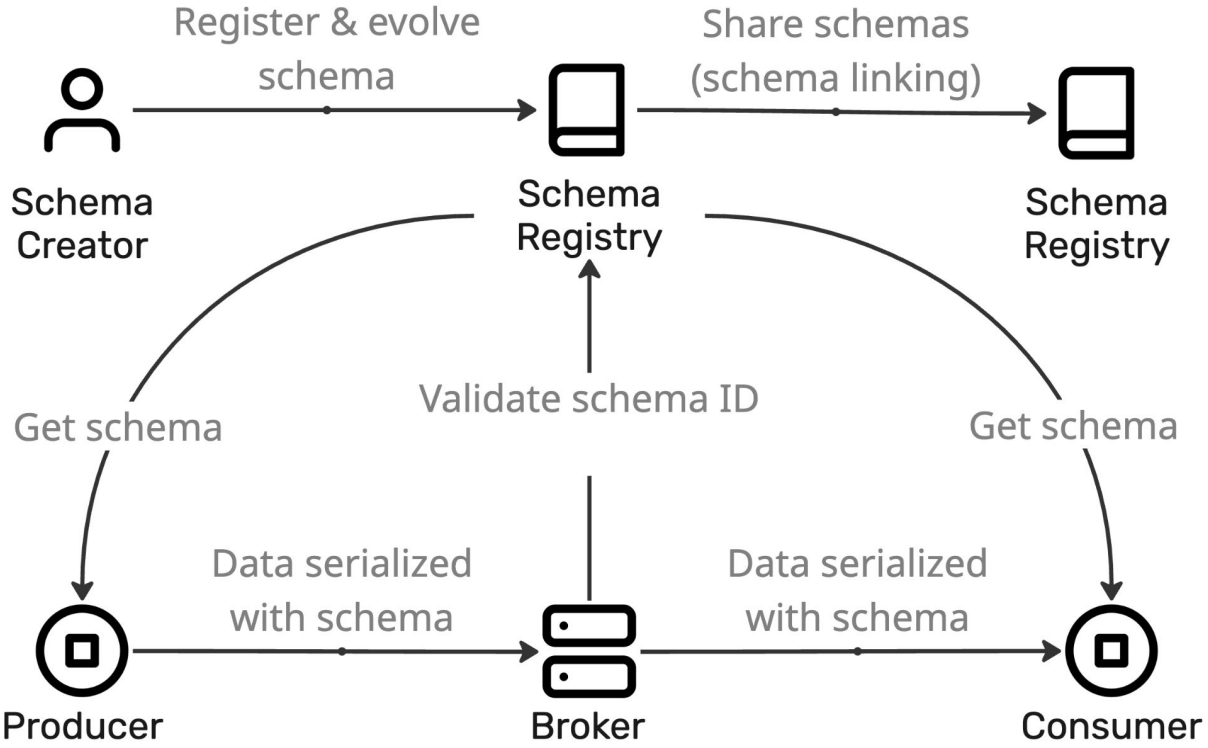
Schema Evolution: Schema Registries



© Sarah Sze, Timelapse, 2023, Guggenheim Museum, New York

A schema registry is just a central place storing different schema definitions.

Schema Registry



Junker, A.; Lazzaretti, F.: Crafting Great APIs using DDD, apress 2025

- Enforcing of a **particular schema** in combination with a broker or proxy
- A **repository** where different schemas are registered, validated, evolved, and checked for interoperability.
- **Management of schemas** using compatibility rules (e.g. **breaking changes**).
- **Data validation** against registered schemas on the producer and consumer side.
- **Provide schema identifiers** with short references instead of using full schema definitions.
- Kafka and UIs like Kafbat help in **debugging,** too

S. Pelluru, A. Chhabria, R. West, and K. Indrasiri, Schema registry in azure event hubs, Jan. 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/eventhubs/schema-registry-concepts (visited on 02/02/2025).

Confluent, Schema registry for confluent platform, 2025. [Online]. Available: https://docs.confluent.io/platform/current/schemaregistry/index.html (visited on 02/02/2025).

Schema Registry Binding

```yaml
asyncapi: 3.0.0
info:
  title: Example with Schema Registry
  version: 0.1.0
servers:
  kafkaServer:
    host: kafkacluster:9092
    protocol: kafka
    bindings:
      kafka:
        schemaRegistryUrl:
          'http://localhost:8080/apis/registry/'
        schemaRegistryVendor: 'apicurio'
        bindingVersion: '0.5.0'
...
```

```yaml
components:
  messages:
    userSignedUp:
      bindings:
        kafka:
          bindingVersion: '0.5.0'
          schemaIdLocation: 'payload'
          schemaIdPayloadEncoding: 'apicurio-new'
          schemaLookupStrategy: 'TopicIdStrategy'
      payload:
        schema:
        ...
```

AsyncAPI does not support directly multiple schema versions, only as binding to a schema registry.

A. Nema, S. Gupta, A. Goyal, and A. Pahti, Message validation, May 2024.
[Online]. Available:
https://www.asyncapi.com/docs/guides/message-validation
(visited on 02/07/2025).

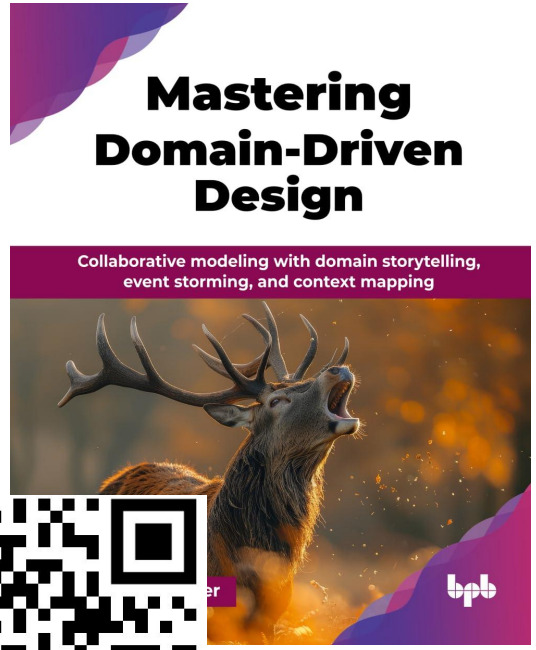
Conclusion

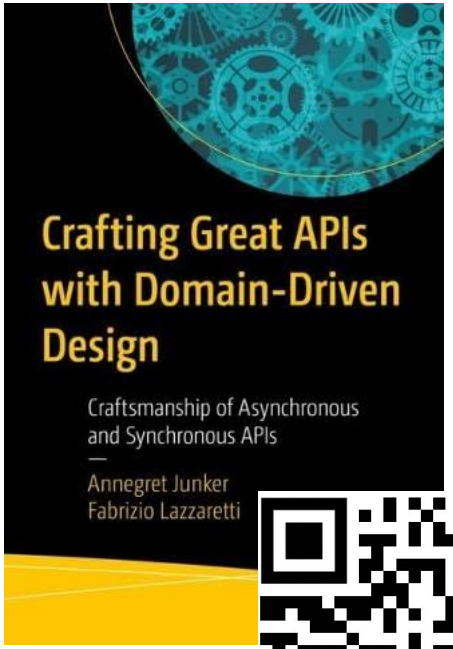
- Avro has a clear evolution strategy
- Avro falls back to a default value, if a wrong format is provided.
- JSON usage offers grey zones in communication

Thank you for listening

Contact Annegret
https://github.com/Grinseteddy
https://www.linkedin.com
/in/dr-annegret-junker-141a99a4/

Contact Fabrizio
https://github.com/Lazzaretti
https://www.linkedin.com
/in/fabrizio-lazzaretti/





THANK YOU!

codecentric WAVESTONE