AVRO - OVERVIEW

Data serialization is a mechanism to translate data in computer environment *likememorybuffer*, *datastructuresorobjectstate* into binary or textual form that can be transported over network or stored in some persistent storage media.

Java and Hadoop provides serialization APIs, which are java based, but Avro is not only language independent but also it is schema-based. We shall explore more difference among them in coming chapter.

What is Avro?

Apache Avro is a language-neutral data serialization system. It was developed by Doug Cutting, the father of Hadoop. Since Hadoop writable classes lack language portability, Avro becomes quite helpful, as it deals with data formats that can be processed by multiple languages. Avro is a preferred tool to serialize data in Hadoop.

Avro has a schema-based system. A language-independent schema is associated with its read and write operations. Avro serializes the data which has a built-in schema. Avro serializes the data into a compact binary format, which can be deserialized by any application.

Avro uses JSON format to declare the data structures. Presently, it supports languages such as Java, C, C++, C#, Python, and Ruby.

Avro Schemas

Avro depends heavily on its **schema**. It allows every data to be written with no prior knowledge of the schema. It serializes fast and the resulting serialized data is lesser in size. Schema is stored along with the Avro data in a file for any further processing.

In RPC, the client and the server exchange schemas during the connection. This exchange helps in the communication between same named fields, missing fields, extra fields, etc.

Avro schemas are defined with JSON that simplifies its implementation in languages with JSON libraries.

Like Avro, there are other serialization mechanisms in Hadoop such as **Sequence Files, Protocol Buffers,** and **Thrift**.

Thrift & Protocol Buffers Vs. Avro

Thrift and **Protocol Buffers** are the most competent libraries with Avro. Avro differs from these frameworks in the following ways −

- Avro supports both dynamic and static types as per the requirement. Protocol Buffers and Thrift use Interface Definition Languages (**IDLs**) to specify schemas and their types. These IDLs are used to generate code for serialization and deserialization.
- Avro is built in the Hadoop ecosystem. Thrift and Protocol Buffers are not built in Hadoop ecosystem.

Unlike Thrift and Protocol Buffer, Avro's schema definition is in JSON and not in any proprietary IDL.

Property	Avro	Thrift & Protocol Buffer
Dynamic schema	Yes	No
Built into Hadoop	Yes	No
Schema in JSON	Yes	No

No need to compile	Yes	No
No need to declare IDs	Yes	No
Bleeding edge	Yes	No

Features of Avro

Listed below are some of the prominent features of Avro −

- Avro is a **language-neutral** data serialization system.
- It can be processed by many languages currently C, C++, C#, Java, Python, and Ruby.
- Avro creates binary structured format that is both **compressible** and **splittable**. Hence it can be efficiently used as the input to Hadoop MapReduce jobs.
- Avro provides **rich data structures**. For example, you can create a record that contains an array, an enumerated type, and a sub record. These datatypes can be created in any language, can be processed in Hadoop, and the results can be fed to a third language.
- Avro **schemas** defined in **JSON**, facilitate implementation in the languages that already have JSON libraries.
- Avro creates a self-describing file named *Avro Data File,* in which it stores data along with its schema in the metadata section.
- Avro is also used in Remote Procedure Calls *RPCs*. During RPC, client and server exchange schemas in the connection handshake.

How to use Avro?

To use Avro, you need to follow the given workflow −

- **Step 1** − Create schemas. Here you need to design Avro schema according to your data.
- **Step 2** − Read the schemas into your program. It is done in two ways −
  - **By Generating a Class Corresponding to Schema** − Compile the schema using Avro. This generates a class file corresponding to the schema
  - **By Using Parsers Library** − You can directly read the schema using parsers library.
- **Step 3** − Serialize the data using the serialization API provided for Avro, which is found in the **package org.apache.avro.specific**.
- **Step 4** − Deserialize the data using deserialization API provided for Avro, which is found in the **package org.apache.avro.specific.**