✅ Kafka Day 11 – Schema Registry, Avro & Schema Evolution (with Answers)

---

1. What is Schema Registry and why is it needed?

Answer:

Schema Registry is a centralized service that stores and manages message schemas used by Kafka producers and consumers.
It ensures **schema compatibility**, tracks versions, and prevents breaking changes during schema evolution.

Without it, producers and consumers might deserialize messages incorrectly due to mismatched schemas, leading to data corruption or crashes.

---

2. Why is Avro used in Kafka instead of JSON or Protobuf?

Answer:

Avro is compact, faster to serialize/deserialize, and supports **built-in schema evolution**.
Unlike JSON (verbose) or Protobuf (less friendly with evolution), Avro embeds a **schema ID** in the message and stores the schema separately in the registry.

It integrates well with big data tools (Spark, Flink) and is highly suited for **streaming systems** with changing schemas.

---

3. How does Avro + Schema Registry work together in Kafka?

Answer:

When a producer sends an Avro message:

1. It serializes data using an Avro schema.
2. The schema is registered (if new), and a **schema ID** is returned.
3. The Kafka message includes this schema ID in a **5-byte prefix**.

On the consumer side:

- The deserializer reads the schema ID from the message.
- It fetches the schema from Schema Registry.
- Then it deserializes the message using that schema.

This makes the system schema-aware and version-safe.

---

4. What are the types of schema compatibility in Schema Registry?

Type	Definition
Backward	New schema can read old data
Forward	Old schema can read new data
Full	Both backward and forward
None	No compatibility check enforced

Answer:

> In production, I use **backward compatibility** because producers often evolve faster than consumers.
> This ensures that consumers can still read data even after new fields are added by producers.

---

5. What changes are considered compatible/incompatible in Avro?

Change	Backward Compatible?	Reason
Add field with default	✅ Yes	Consumer won't break
Remove field	✅ If unused	Safe if old field not needed
Rename field	❌ No	Avro uses field name as key
Change type (int→long)	⚠️ Sometimes	Compatible only in widening conversions

Answer:

> I ensure new fields have defaults to keep compatibility.
> I avoid renaming or changing types unless the schema is being reset.

---

6. How do you register or evolve a schema?

Answer:

> When using KafkaAvroSerializer, the producer **auto-registers** the schema on first use.
> You can also register manually via:

bash
CopyEdit

```
POST /subjects/<topic>-value/versions
```

> Schema Registry handles versioning and validates compatibility against previous versions before allowing new ones.

---

7. What happens if the producer sends a message with a newer schema but the consumer uses an older version?

Answer:

> If compatibility mode is set to **backward**, the consumer will **still be able to read the message** — as long as:
>
> - The new schema only added fields (with defaults).
> - No renamed or removed fields.
>
> This is **why schema evolution rules are enforced by Schema Registry** — to prevent runtime breakages.

---

8. Real-World Example

Q: You added a new field `region` to the Avro schema. Some consumers haven't updated yet. What will you do?

Answer:

> I'll add the field `region` with a default value like:

json
CopyEdit

```
{ "name": "region", "type": "string", "default": "" }
```

> Then I'll set the compatibility level to **BACKWARD** to ensure old consumers still work.
> This allows safe schema evolution without breaking consumers.

---

9. Tools for Working with Avro + Schema Registry

Answer:

- **KafkaAvroSerializer** / **KafkaAvroDeserializer** (for Java)
- **Confluent Schema Registry** or **Karapace** (open-source)
- **Avro-tools** to convert/view `.avsc` files
- **REST API** for managing schemas
- Integrated with **Spark Structured Streaming**, **Flink**, and **Kafka Streams**

Would you like a **mock interview for Day 11**, or shall I send this as a **printable PDF cheat sheet**?

4o

You said:

So you tell me one thing, from the upstream, the messages will be coming in a serialized manner. So it will first compress it and then it will send it to the partition. There again serialization will happen, for what?

ChatGPT said:

Great observation — and here's the **clear breakdown** of what happens and **why serialization doesn't happen twice**:

Message Flow from Producer to Broker – Serialization & Compression

Step-by-step:

1. **Producer receives raw data** (e.g., a Java/JSON/Python object).
2. **Serialization happens first**:
    - The message is converted into **binary format** (e.g., Avro/JSON/Protobuf).
    - If Avro is used, schema ID is prepended.
    - This step makes it **ready for transport/storage**.
3. **Then compression happens** (if enabled):
    - Kafka compresses **the entire batch** of already-serialized messages.
    - Compression is applied per **record batch**, not per message.
    - Config: `compression.type = gzip/snappy/lz4/zstd`
4. **Compressed binary batch is sent to the partition (broker)**.
    - Broker stores the bytes **as-is**. No re-serialization.
    - **Kafka brokers do not deserialize or serialize again** — they just store the payload.

So, to clarify your question:

> **Does serialization happen twice?**

No.

- **Only the producer serializes.**
- Kafka brokers store it as-is.
- **Consumers** then **deserialize** the message using the schema ID to fetch schema from the registry.

Where Each Operation Happens

Step	Who Does It	Purpose
Serialization	Producer	Convert object to bytes
Compression	Kafka client (Producer)	Reduce payload size
Deserialization	Consumer	Convert bytes back to object
Schema Resolution	Consumer via Schema Registry	To interpret the Avro-encoded message