Sumo Ingestion MVP (Kafka → S3)

This starter packs a local stack to receive webhooks, validate against JSON Schemas, publish to Kafka, and land raw events into an S3-compatible bucket (MinIO). It uses Redpanda (Kafka-API compatible) for simplicity.

---

1) Folder layout

```
sumo-ingestion/
├─ docker-compose.yml
├─ .env
├─ app/
│  ├─ requirements.txt
│  ├─ webhook.py
│  └─ schemas/
│     ├─ match_result.schema.json
│     └─ user_prediction.schema.json
└─ spark/
   └─ kafka_to_s3.py
```

---

2) docker-compose.yml

```
version: "3.9"
services:
  redpanda:
    image: redpandadata/redpanda:v24.1.9
    command:
      - redpanda start
      - --mode=dev-container
      - --overprovisioned
      - --smp=1
      - --memory=1G
      - --reserve-memory=0M
      - --node-id=0
      - --check=false
      - --pandaproxy-addr=0.0.0.0:8082
      - --kafka-addr=PLAINTEXT://0.0.0.0:9092,OUTSIDE://0.0.0.0:19092
      - --advertise-kafka-addr=PLAINTEXT://redpanda:9092,OUTSIDE://localhost:
19092
    ports:
```

```
      - 9092:9092
      - 19092:19092
      - 9644:9644
      - 8082:8082
    volumes:
      - redpanda-data:/var/lib/redpanda/data

  console:
    image: redpandadata/console:latest
    environment:
      - KAFKA_BROKERS=redpanda:9092
    ports:
      - 8080:8080

  minio:
    image: minio/minio:latest
    command: server /data --console-address ":9001"
    environment:
      MINIO_ROOT_USER: ${MINIO_ROOT_USER}
      MINIO_ROOT_PASSWORD: ${MINIO_ROOT_PASSWORD}
    ports:
      - 9000:9000   # S3 API
      - 9001:9001   # MinIO Console
    volumes:
      - minio-data:/data

  create-bucket:
    image: minio/mc:latest
    depends_on:
      - minio
    entrypoint: ["/bin/sh","-c"]
    environment:
      MINIO_ROOT_USER: ${MINIO_ROOT_USER}
      MINIO_ROOT_PASSWORD: ${MINIO_ROOT_PASSWORD}
    command: >
      "mc alias set local http://minio:9000 $MINIO_ROOT_USER
$MINIO_ROOT_PASSWORD &&
       mc mb -p local/sumo-lake &&
       mc policy set public local/sumo-lake &&
       sleep 2 && echo 'Bucket ready'"

  webhook:
    build:
      context: ./app
      dockerfile: Dockerfile
    environment:
      KAFKA_BOOTSTRAP: redpanda:9092
      WEBHOOK_SECRET: ${WEBHOOK_SECRET}
```

```
    ports:
      - 5000:5000
    depends_on:
      - redpanda

volumes:
  redpanda-data: {}
  minio-data: {}
```

---

3) .env (create in project root)

```
MINIO_ROOT_USER=minioadmin
MINIO_ROOT_PASSWORD=minioadmin
WEBHOOK_SECRET=supersecret_shared_hmac
```

---

4) app/Dockerfile

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 5000
CMD ["python","webhook.py"]
```

---

5) app/requirements.txt

```
Flask==3.0.3
jsonschema==4.23.0
confluent-kafka==2.5.3
python-dotenv==1.0.1
```

---

6) app/schemas/match_result.schema.json

```
{
  "$id": "https://yourorg/schemas/sumo/match_result.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "match_result",
  "type": "object",
  "additionalProperties": false,
  "required": [
    "event","schema_version","match_id","tournament_id","division","day",
    "rikishi1_id","rikishi2_id","winner_id","kimarite","announced_at",
    "ingested_at","idempotency_key"
  ],
  "properties": {
    "event": {"const": "match_result"},
    "schema_version": {"type": "string"},
    "match_id": {"type": "string"},
    "tournament_id": {"type": "string"},
    "division": {"type": "string"},
    "day": {"type": "integer", "minimum": 1, "maximum": 15},
    "rikishi1_id": {"type": "string"},
    "rikishi2_id": {"type": "string"},
    "winner_id": {"type": "string"},
    "kimarite": {"type": "string"},
    "announced_at": {"type": "string", "format": "date-time"},
    "ingested_at": {"type": "string", "format": "date-time"},
    "source": {"type": "string"},
    "idempotency_key": {"type": "string", "minLength": 16}
  }
}
```

---

7) app/schemas/user_prediction.schema.json

```
{
  "$id": "https://yourorg/schemas/sumo/user_prediction.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "user_prediction",
  "type": "object",
  "additionalProperties": false,
  "required": [
    "event","schema_version","prediction_id","user_id","match_id",
    "predicted_winner_id","created_at","ingested_at","idempotency_key"
  ],
```

```
  "properties": {
    "event": {"const": "user_prediction"},
    "schema_version": {"type": "string"},
    "prediction_id": {"type": "string"},
    "user_id": {"type": "string"},
    "match_id": {"type": "string"},
    "predicted_winner_id": {"type": "string"},
    "created_at": {"type": "string", "format": "date-time"},
    "ingested_at": {"type": "string", "format": "date-time"},
    "client": {"type": "string"},
    "idempotency_key": {"type": "string", "minLength": 8}
  }
}
```

---

8) app/webhook.py

```python
import os, hmac, hashlib, json
from datetime import datetime, timezone
from flask import Flask, request, jsonify
from confluent_kafka import Producer
from jsonschema import Draft202012Validator, exceptions as js_exc

# Load schemas
with open("schemas/match_result.schema.json") as f:
    MATCH_SCHEMA = json.load(f)
with open("schemas/user_prediction.schema.json") as f:
    PRED_SCHEMA = json.load(f)

SCHEMAS = {
    "match_result": Draft202012Validator(MATCH_SCHEMA),
    "user_prediction": Draft202012Validator(PRED_SCHEMA),
}

KAFKA_BOOTSTRAP = os.getenv("KAFKA_BOOTSTRAP", "localhost:19092")
WEBHOOK_SECRET = os.getenv("WEBHOOK_SECRET", "")

producer = Producer({"bootstrap.servers": KAFKA_BOOTSTRAP})

app = Flask(__name__)

def verify_signature(secret: str, body: bytes, their_sig: str) -> bool:
    if not secret:
        return True  # dev mode
    digest = hmac.new(secret.encode(), body, hashlib.sha256).hexdigest()
```

```
    return hmac.compare_digest(digest, (their_sig or ""))

@app.post("/sumo-webhook")
def handle_webhook():
    raw = request.get_data()
    sig = request.headers.get("X-Sumo-Signature")
    if not verify_signature(WEBHOOK_SECRET, raw, sig):
        return jsonify({"error": "invalid signature"}), 401

    try:
        payload = json.loads(raw)
    except json.JSONDecodeError:
        return jsonify({"error": "invalid json"}), 400

    event_type = payload.get("event")
    if event_type not in SCHEMAS:
        return jsonify({"error": "unknown event"}), 400

    # Auto-fill ingestion timestamp if missing
    payload.setdefault("ingested_at", datetime.now(timezone.utc).isoformat())

    try:
        SCHEMAS[event_type].validate(payload)
    except js_exc.ValidationError as e:
        return jsonify({"error": "schema validation failed", "detail":
e.message}), 400

    # Choose topic and key
    if event_type == "match_result":
        topic = "sumo.matches"
        key = payload.get("match_id", "")
    else:
        topic = "sumo.predictions"
        key = payload.get("match_id", "")  # per-match partitioning

    producer.produce(topic, key=key, value=json.dumps(payload).encode("utf-8"))
    producer.flush()

    return jsonify({"status": "ok"}), 200

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

---

9) spark/kafka_to_s3.py (Structured Streaming to MinIO/S3)

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, from_json
from pyspark.sql.types import StringType
import json

# Minimal dynamic schema pass-through (treat value as JSON string → column
"json")
# For production, consider explicit schemas per topic.

def main():
    spark = (SparkSession.builder
             .appName("KafkaToS3Raw")
             .config("spark.sql.shuffle.partitions", "4")
             .getOrCreate())

    bootstrap = "redpanda:9092"
    s3_endpoint = "http://minio:9000"  # for local MinIO

    # Configure S3/MinIO
    hadoop_conf = spark._jsc.hadoopConfiguration()
    hadoop_conf.set("fs.s3a.endpoint", s3_endpoint)
    hadoop_conf.set("fs.s3a.access.key", "minioadmin")
    hadoop_conf.set("fs.s3a.secret.key", "minioadmin")
    hadoop_conf.set("fs.s3a.path.style.access", "true")
    hadoop_conf.set("fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem")

    def make_stream(topic, prefix):
        df = (spark.readStream
              .format("kafka")
              .option("kafka.bootstrap.servers", bootstrap)
              .option("subscribe", topic)
              .option("startingOffsets", "latest")
              .load())

        json_str = df.selectExpr("CAST(value AS STRING) as json")

        # Write raw JSON as parquet in date-partitioned folders
        query = (json_str
                 .writeStream
                 .format("json")
                 .option("path", f"s3a://sumo-lake/raw/{prefix}")
                 .option("checkpointLocation", f"/tmp/checkpoints/{prefix}")
                 .trigger(processingTime="30 seconds")
                 .start())
```

```
        return query

    q1 = make_stream("sumo.matches", "match_result")
    q2 = make_stream("sumo.predictions", "user_prediction")

    q1.awaitTermination()
    q2.awaitTermination()

if __name__ == "__main__":
    main()
```

Note: We write **raw JSON files** to MinIO for clarity. You can switch to `format("parquet")` later; just ensure you parse and structure columns first.

---

10) Run it

a) Start the stack

```
docker compose up -d --build
```

Open Redpanda Console: http://localhost:8080 Open MinIO Console: http://localhost:9001 (use creds from `.env`).

b) Create topics (via Console UI or CLI inside container)

```
# Using rpk inside the redpanda container
docker compose exec redpanda rpk topic create sumo.matches sumo.predictions
```

c) Send a test webhook

```
curl -X POST http://localhost:5000/sumo-webhook \
  -H 'Content-Type: application/json' \
  -d '{
    "event":"match_result",
    "schema_version":"v1.0.0",
    "match_id":"TEST-001",
    "tournament_id":"JUL-2025",
    "division":"makuuchi",
    "day":1,
    "rikishi1_id":"terunofuji",
    "rikishi2_id":"hoshoryu",
    "winner_id":"terunofuji",
```

```
    "kimarite":"yorikiri",
    "announced_at":"2025-07-10T03:15:22Z",
    "ingested_at":"2025-07-10T03:15:25Z",
    "idempotency_key":"match_result|TEST-001|2025-07-10T03:15:22Z"
  }'
```

You should see the event in topic **sumo.matches** (Console) and, once the Spark job is running, files appear under `sumo-lake/raw/match_result` in MinIO.

d) Launch Spark job (local containerized example)

Use your preferred Spark runtime; for quick local dev you can run this **from your host** if Spark is installed, or package into a container. Example host command:

```
spark-submit \
  --packages org.apache.spark:spark-sql-
kafka-0-10_2.12:3.5.1,org.apache.hadoop:hadoop-aws:3.3.4 \
  spark/kafka_to_s3.py
```

If running inside Docker, ensure the Spark image can reach `redpanda:9092` and `minio:9000` on the compose network (or adjust to `localhost` with port mappings).

---

11) Next steps

- Switch raw writer to **Parquet** and partition by `event_date=YYYY-MM-DD`.
- Add **Great Expectations** checks on landed data.
- Wire **Airflow** to run the Spark job in a simple DAG.
- Add HMAC header generation to your real Sumo source; for local tests, the app accepts missing secrets. ```