**PostgreSQL to Text File via Kafka - Complete Project**

**Project Architecture**

PostgreSQL → Python Producer → Kafka → Python Consumer → Text Files

**Project Structure**

kafka-postgres-project/

├── docker-compose.yml

├── producer/

│ ├── producer.py

│ ├── requirements.txt

│ └── Dockerfile

├── consumer/

│ ├── consumer.py

│ ├── requirements.txt

│ └── Dockerfile

├── postgres/

│ └── init.sql

└── output/

└── (text files will be generated here)

**Step 1: Docker Compose Setup**

Create docker-compose.yml:

version: '3.8'

services:

# PostgreSQL Database

postgres:

image: postgres:15

container\_name: postgres

environment:

POSTGRES\_DB: testdb

POSTGRES\_USER: postgres

POSTGRES\_PASSWORD: password

ports:

- "5432:5432"

volumes:

- ./postgres/init.sql:/docker-entrypoint-initdb.d/init.sql

- postgres\_data:/var/lib/postgresql/data

restart: unless-stopped

# Kafka Broker

kafka:

image: confluentinc/cp-kafka:7.6.0

container\_name: kafka

ports:

- "9092:9092"

- "29092:29092"

environment:

KAFKA\_NODE\_ID: 1

KAFKA\_PROCESS\_ROLES: broker,controller

KAFKA\_LISTENERS: PLAINTEXT://0.0.0.0:9092,CONTROLLER://0.0.0.0:9093,PLAINTEXT\_HOST://0.0.0.0:29092

KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://kafka:9092,PLAINTEXT\_HOST://localhost:29092

KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: CONTROLLER:PLAINTEXT,PLAINTEXT:PLAINTEXT,PLAINTEXT\_HOST:PLAINTEXT

KAFKA\_CONTROLLER\_QUORUM\_VOTERS: 1@kafka:9093

KAFKA\_CONTROLLER\_LISTENER\_NAMES: CONTROLLER

KAFKA\_INTER\_BROKER\_LISTENER\_NAME: PLAINTEXT

KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1

KAFKA\_TRANSACTION\_STATE\_LOG\_REPLICATION\_FACTOR: 1

KAFKA\_TRANSACTION\_STATE\_LOG\_MIN\_ISR: 1

KAFKA\_GROUP\_INITIAL\_REBALANCE\_DELAY\_MS: 0

CLUSTER\_ID: z1IhO92KQeO7JkTnsPq4PA

restart: unless-stopped

depends\_on:

- postgres

# Data Producer

producer:

build: ./producer

container\_name: producer

depends\_on:

- postgres

- kafka

environment:

POSTGRES\_HOST: postgres

POSTGRES\_PORT: 5432

POSTGRES\_DB: testdb

POSTGRES\_USER: postgres

POSTGRES\_PASSWORD: password

KAFKA\_BROKER: kafka:9092

KAFKA\_TOPIC: postgres-data

restart: unless-stopped

# Data Consumer

consumer:

build: ./consumer

container\_name: consumer

depends\_on:

- kafka

environment:

KAFKA\_BROKER: kafka:9092

KAFKA\_TOPIC: postgres-data

KAFKA\_GROUP: file-writer-group

volumes:

- ./output:/app/output

restart: unless-stopped

volumes:

postgres\_data:

**Step 2: PostgreSQL Setup**

Create postgres/init.sql:

-- Create sample table

CREATE TABLE IF NOT EXISTS users (

id SERIAL PRIMARY KEY,

name VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

age INTEGER,

city VARCHAR(50),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- Insert sample data

INSERT INTO users (name, email, age, city) VALUES

('John Doe', 'john@example.com', 30, 'New York'),

('Jane Smith', 'jane@example.com', 25, 'Los Angeles'),

('Mike Johnson', 'mike@example.com', 35, 'Chicago'),

('Sarah Wilson', 'sarah@example.com', 28, 'Houston'),

('David Brown', 'david@example.com', 32, 'Phoenix');

-- Create products table

CREATE TABLE IF NOT EXISTS products (

id SERIAL PRIMARY KEY,

name VARCHAR(100) NOT NULL,

price DECIMAL(10,2),

category VARCHAR(50),

stock INTEGER DEFAULT 0,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- Insert sample products

INSERT INTO products (name, price, category, stock) VALUES

('Laptop', 999.99, 'Electronics', 50),

('Smartphone', 699.99, 'Electronics', 100),

('Desk Chair', 149.99, 'Furniture', 25),

('Coffee Mug', 12.99, 'Kitchen', 200),

('Notebook', 5.99, 'Stationery', 500);

**Step 3: Producer Application**

Create producer/requirements.txt:

kafka-python==2.0.2

psycopg2-binary==2.9.9

python-dotenv==1.0.0

Create producer/Dockerfile:

FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY producer.py .

CMD ["python", "producer.py"]

Create producer/producer.py:

import json

import time

import os

import logging

from kafka import KafkaProducer

import psycopg2

from psycopg2.extras import RealDictCursor

# Configure logging

logging.basicConfig(level=logging.INFO)

logger = logging.getLogger(\_\_name\_\_)

class PostgreSQLKafkaProducer:

def \_\_init\_\_(self):

self.kafka\_broker = os.getenv('KAFKA\_BROKER', 'kafka:9092')

self.kafka\_topic = os.getenv('KAFKA\_TOPIC', 'postgres-data')

self.pg\_host = os.getenv('POSTGRES\_HOST', 'postgres')

self.pg\_port = os.getenv('POSTGRES\_PORT', '5432')

self.pg\_db = os.getenv('POSTGRES\_DB', 'testdb')

self.pg\_user = os.getenv('POSTGRES\_USER', 'postgres')

self.pg\_password = os.getenv('POSTGRES\_PASSWORD', 'password')

self.producer = None

self.pg\_conn = None

def connect\_kafka(self):

"""Connect to Kafka"""

max\_retries = 10

for attempt in range(max\_retries):

try:

self.producer = KafkaProducer(

bootstrap\_servers=[self.kafka\_broker],

value\_serializer=lambda v: json.dumps(v).encode('utf-8'),

key\_serializer=lambda v: v.encode('utf-8') if v else None

)

logger.info("Connected to Kafka successfully")

return True

except Exception as e:

logger.error(f"Kafka connection attempt {attempt + 1} failed: {e}")

time.sleep(5)

return False

def connect\_postgres(self):

"""Connect to PostgreSQL"""

max\_retries = 10

for attempt in range(max\_retries):

try:

self.pg\_conn = psycopg2.connect(

host=self.pg\_host,

port=self.pg\_port,

database=self.pg\_db,

user=self.pg\_user,

password=self.pg\_password

)

logger.info("Connected to PostgreSQL successfully")

return True

except Exception as e:

logger.error(f"PostgreSQL connection attempt {attempt + 1} failed: {e}")

time.sleep(5)

return False

def fetch\_and\_send\_data(self, table\_name, batch\_size=10):

"""Fetch data from PostgreSQL and send to Kafka"""

try:

with self.pg\_conn.cursor(cursor\_factory=RealDictCursor) as cursor:

cursor.execute(f"SELECT COUNT(\*) FROM {table\_name}")

total\_records = cursor.fetchone()[0]

logger.info(f"Found {total\_records} records in {table\_name}")

offset = 0

while offset < total\_records:

cursor.execute(

f"SELECT \* FROM {table\_name} ORDER BY id LIMIT %s OFFSET %s",

(batch\_size, offset)

)

rows = cursor.fetchall()

if not rows:

break

for row in rows:

# Convert to dict and handle datetime objects

data = dict(row)

# Convert datetime to string

for key, value in data.items():

if hasattr(value, 'isoformat'):

data[key] = value.isoformat()

# Add metadata

message = {

'table': table\_name,

'data': data,

'timestamp': time.time()

}

# Send to Kafka

key = f"{table\_name}\_{data['id']}"

self.producer.send(self.kafka\_topic, value=message, key=key)

logger.info(f"Sent record {data['id']} from {table\_name}")

offset += batch\_size

time.sleep(1) # Small delay between batches

self.producer.flush()

logger.info(f"Finished processing {table\_name}")

except Exception as e:

logger.error(f"Error fetching data from {table\_name}: {e}")

def run(self):

"""Main execution loop"""

logger.info("Starting PostgreSQL to Kafka Producer...")

# Connect to services

if not self.connect\_kafka():

logger.error("Failed to connect to Kafka")

return

if not self.connect\_postgres():

logger.error("Failed to connect to PostgreSQL")

return

# Create topic if it doesn't exist

try:

from kafka.admin import KafkaAdminClient, NewTopic

admin = KafkaAdminClient(bootstrap\_servers=[self.kafka\_broker])

topic = NewTopic(name=self.kafka\_topic, num\_partitions=1, replication\_factor=1)

admin.create\_topics([topic])

logger.info(f"Created topic {self.kafka\_topic}")

except Exception as e:

logger.info(f"Topic {self.kafka\_topic} may already exist: {e}")

# Extract data from tables

tables = ['users', 'products']

while True:

for table in tables:

logger.info(f"Processing table: {table}")

self.fetch\_and\_send\_data(table)

logger.info("Completed one cycle, waiting 30 seconds...")

time.sleep(30) # Wait before next extraction cycle

if \_\_name\_\_ == "\_\_main\_\_":

producer = PostgreSQLKafkaProducer()

producer.run()

**Step 4: Consumer Application**

Create consumer/requirements.txt:

kafka-python==2.0.2

python-dotenv==1.0.0

Create consumer/Dockerfile:

FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY consumer.py .

# Create output directory

RUN mkdir -p /app/output

CMD ["python", "consumer.py"]

Create consumer/consumer.py:

import json

import os

import time

import logging

from datetime import datetime

from kafka import KafkaConsumer

# Configure logging

logging.basicConfig(level=logging.INFO)

logger = logging.getLogger(\_\_name\_\_)

class KafkaFileWriter:

def \_\_init\_\_(self):

self.kafka\_broker = os.getenv('KAFKA\_BROKER', 'kafka:9092')

self.kafka\_topic = os.getenv('KAFKA\_TOPIC', 'postgres-data')

self.kafka\_group = os.getenv('KAFKA\_GROUP', 'file-writer-group')

self.output\_dir = '/app/output'

# Ensure output directory exists

os.makedirs(self.output\_dir, exist\_ok=True)

self.consumer = None

self.file\_handles = {}

def connect\_kafka(self):

"""Connect to Kafka consumer"""

max\_retries = 10

for attempt in range(max\_retries):

try:

self.consumer = KafkaConsumer(

self.kafka\_topic,

bootstrap\_servers=[self.kafka\_broker],

group\_id=self.kafka\_group,

value\_deserializer=lambda m: json.loads(m.decode('utf-8')),

key\_deserializer=lambda m: m.decode('utf-8') if m else None,

auto\_offset\_reset='latest', # Start from latest messages

enable\_auto\_commit=True,

consumer\_timeout\_ms=1000

)

logger.info("Connected to Kafka consumer successfully")

return True

except Exception as e:

logger.error(f"Kafka consumer connection attempt {attempt + 1} failed: {e}")

time.sleep(5)

return False

def get\_file\_handle(self, table\_name):

"""Get or create file handle for a table"""

if table\_name not in self.file\_handles:

timestamp = datetime.now().strftime("%Y%m%d\_%H%M%S")

filename = f"{table\_name}\_{timestamp}.txt"

filepath = os.path.join(self.output\_dir, filename)

self.file\_handles[table\_name] = {

'handle': open(filepath, 'w'),

'filename': filename,

'record\_count': 0

}

# Write header

self.file\_handles[table\_name]['handle'].write(f"=== {table\_name.upper()} DATA ===\\n")

self.file\_handles[table\_name]['handle'].write(f"Generated at: {datetime.now().isoformat()}\\n")

self.file\_handles[table\_name]['handle'].write("=" \* 50 + "\\n\\n")

logger.info(f"Created new file: {filename}")

return self.file\_handles[table\_name]

def write\_record\_to\_file(self, message):

"""Write a single record to appropriate text file"""

try:

table\_name = message['table']

data = message['data']

timestamp = message['timestamp']

file\_info = self.get\_file\_handle(table\_name)

# Write record to file

file\_handle = file\_info['handle']

file\_handle.write(f"Record ID: {data.get('id', 'N/A')}\\n")

file\_handle.write(f"Timestamp: {datetime.fromtimestamp(timestamp).isoformat()}\\n")

# Write all fields

for key, value in data.items():

file\_handle.write(f"{key}: {value}\\n")

file\_handle.write("-" \* 30 + "\\n\\n")

file\_handle.flush() # Ensure data is written

file\_info['record\_count'] += 1

logger.info(f"Written record {data.get('id')} to {file\_info['filename']} "

f"(total: {file\_info['record\_count']} records)")

except Exception as e:

logger.error(f"Error writing record to file: {e}")

def rotate\_files\_if\_needed(self):

"""Rotate files if they get too large"""

max\_records = 100 # Rotate after 100 records

for table\_name, file\_info in list(self.file\_handles.items()):

if file\_info['record\_count'] >= max\_records:

logger.info(f"Rotating file for table {table\_name}")

# Close current file

file\_info['handle'].close()

# Remove from handles (will create new one on next write)

del self.file\_handles[table\_name]

def run(self):

"""Main consumer loop"""

logger.info("Starting Kafka to File Consumer...")

if not self.connect\_kafka():

logger.error("Failed to connect to Kafka")

return

logger.info(f"Consuming messages from topic: {self.kafka\_topic}")

logger.info(f"Output directory: {self.output\_dir}")

try:

while True:

# Poll for messages

message\_batch = self.consumer.poll(timeout\_ms=1000)

if message\_batch:

for topic\_partition, messages in message\_batch.items():

for message in messages:

try:

self.write\_record\_to\_file(message.value)

except Exception as e:

logger.error(f"Error processing message: {e}")

# Check if files need rotation

self.rotate\_files\_if\_needed()

except KeyboardInterrupt:

logger.info("Consumer interrupted by user")

except Exception as e:

logger.error(f"Consumer error: {e}")

finally:

# Clean up file handles

for file\_info in self.file\_handles.values():

file\_info['handle'].close()

if self.consumer:

self.consumer.close()

if \_\_name\_\_ == "\_\_main\_\_":

consumer = KafkaFileWriter()

consumer.run()

**Step 5: Running the Project**

**1. Create project structure:**

mkdir kafka-postgres-project

cd kafka-postgres-project

mkdir producer consumer postgres output

**2. Create all the files above in their respective directories**

**3. Start the services:**

# Start all services

docker-compose up -d

# Check if all containers are running

docker ps

# Check logs

docker logs postgres

docker logs kafka

docker logs producer

docker logs consumer

**4. Monitor the output:**

# Watch the output directory

ls -la output/

# View the content of generated files

cat output/users\_\*.txt

cat output/products\_\*.txt

# Follow producer logs

docker logs -f producer

# Follow consumer logs

docker logs -f consumer

**Expected Output**

The system will create text files like:

**users\_20240826\_143022.txt:**

=== USERS DATA ===

Generated at: 2024-08-26T14:30:22

==================================================

Record ID: 1

Timestamp: 2024-08-26T14:30:22.123456

id: 1

name: John Doe

email: john@example.com

age: 30

city: New York

created\_at: 2024-08-26T14:30:22.123456

------------------------------

Record ID: 2

Timestamp: 2024-08-26T14:30:23.234567

id: 2

name: Jane Smith

email: jane@example.com

age: 25

city: Los Angeles

created\_at: 2024-08-26T14:30:23.234567

------------------------------

**Features**

* **Real-time streaming**: Data flows continuously from PostgreSQL → Kafka → Text files
* **Batch processing**: Producer reads data in configurable batches
* **File rotation**: Files rotate after 100 records (configurable)
* **Error handling**: Robust error handling and retries
* **Multiple tables**: Supports multiple PostgreSQL tables
* **Timestamped files**: Each file has creation timestamp
* **Docker containerized**: Fully containerized setup

**Customization**

You can modify:

* Database queries in producer.py
* File format in consumer.py
* Batch sizes and intervals
* Add more tables to extract
* Change output format (JSON, CSV, etc.)

This is a production-ready streaming data pipeline!