# Kafka POC: End-to-End Build & Debugging Journey

Docker + Kafka + Kafka UI + PostgreSQL + Python Microservices (FastAPI + SQLAlchemy)

Rohan Javed

2026-01-09

## Contents

**1 What This Project Builds** **3**

**2 Architecture and Message Flow** **3**

**3 Prerequisites on Windows (VS Code Workflow)** **3**

**4 The Real Journey: Issues Faced and How They Were Solved** **3**
- 4.1 Phase 1: Kafka UI showed no topics . . . 4
- 4.2 Phase 2: Docker Compose confusion (service missing / build context) . . . 4
- 4.3 Phase 3: "Site can't be reached" for the Order Service . . . 5
- 4.4 Phase 4: Postgres connection refused at startup . . . 5
- 4.5 Phase 5: Kafka connection refused . . . 5
- 4.6 Phase 6: PowerShell curl confusion . . . 6
- 4.7 Phase 7: SQLAlchemy session factory became None (SessionLocal bug) . . . 6
- 4.8 Phase 8: Missing topics (UNKNOWN_TOPIC_OR_PARTITION) . . . 6
- 4.9 Phase 9: Notification service crash: missing import . . . 7
- 4.10 Phase 10: Git/GitHub publishing issues . . . 7
- 4.11 Final Proof: Kafka UI shows flowing events . . . 7

**5 Canonical Command Reference (Copy/Paste)** **7**
- 5.1 Build and start . . . 7
- 5.2 See running containers . . . 7
- 5.3 View logs . . . 8
- 5.4 Create topics . . . 8
- 5.5 Test API (PowerShell) . . . 8
- 5.6 Check database . . . 8
- 5.7 Clean restart . . . 8

**6 Complete Source Code (All Files Under services/) 8**
6.1 services/inventory-service/Dockerfile . . . . . 8
6.2 services/inventory-service/app/db.py . . . . . 9
6.3 services/inventory-service/app/kafka_client.py . . . . . 9
6.4 services/inventory-service/app/models.py . . . . . 10
6.5 services/inventory-service/app/worker.py . . . . . 10
6.6 services/inventory-service/requirements.txt . . . . . 11
6.7 services/notification-service/Dockerfile . . . . . 11
6.8 services/notification-service/app/db.py . . . . . 11
6.9 services/notification-service/app/models.py . . . . . 12
6.10 services/notification-service/app/worker.py . . . . . 12
6.11 services/notification-service/requirements.txt . . . . . 13
6.12 services/order-service/Dockerfile . . . . . 13
6.13 services/order-service/app/db.py . . . . . 14
6.14 services/order-service/app/kafka_client.py . . . . . 14
6.15 services/order-service/app/main.py . . . . . 15
6.16 services/order-service/app/models.py . . . . . 16
6.17 services/order-service/requirements.txt . . . . . 16

**7 How to Upload and Compile on Overleaf 16**
7.1 Upload steps . . . . . 16
7.2 If Overleaf shows “Unknown main document” . . . . . 17
7.3 If Overleaf cannot render PDF . . . . . 17

# 1 What This Project Builds

This project is a complete proof-of-concept (POC) for an **event-driven microservices architecture** using Kafka. It includes:

- **Kafka + ZooKeeper** orchestrated with Docker Compose
- **Kafka UI (Provectus)** for topic/message visibility
- **PostgreSQL** for persistence
- **Order Service** (FastAPI REST API) that writes to DB and publishes `order-events`
- **Inventory Service** (Kafka consumer/producer) that consumes `order-events`, writes to DB, and publishes `inventory-events`
- **Notification Service** (Kafka consumer) that consumes `inventory-events`, logs a simulated email/SMS, and stores a notification record

# 2 Architecture and Message Flow

1. Client sends `POST /orders` to Order Service.
2. Order Service inserts a row into `orders` table and produces an event on Kafka topic `order-events`.
3. Inventory Service consumes `order-events`, inserts a reservation into `inventory` table, then produces an event to `inventory-events`.
4. Notification Service consumes `inventory-events`, logs a simulated notification, and inserts a row into `notifications` table.

# 3 Prerequisites on Windows (VS Code Workflow)

- Docker Desktop installed and running
- Git installed (so you can commit and push the project)
- VS Code installed (used for editing + integrated terminal)
- PowerShell used for commands (Windows default)

**Tip:** Always run commands from the project root directory:

```
C:\Users\ASAD\Documents\kafka-poc
```

# 4 The Real Journey: Issues Faced and How They Were Solved

This section documents the exact problems that occurred while building the system and how they were fixed.

## 4.1 Phase 1: Kafka UI showed no topics

**What you saw:** Kafka UI was accessible, but only internal Kafka topic(s) appeared (e.g. `__consumer_offsets`). No app topics existed yet.

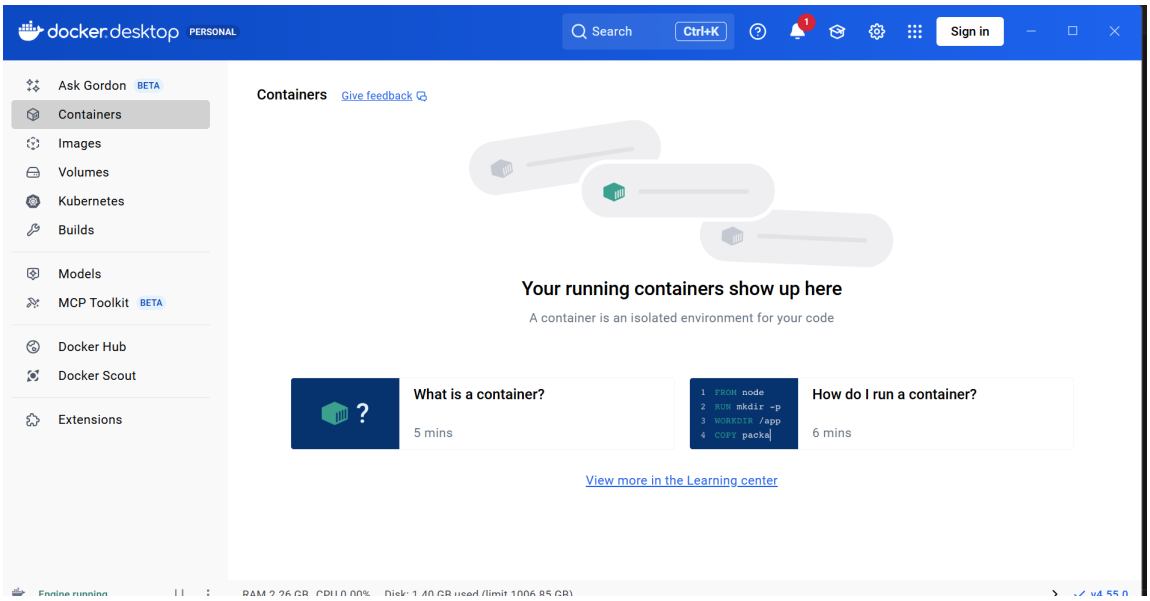


Figure 1: Kafka UI initial view (no application topics).

**Key lesson:** "Kafka container is running" does not mean your application topics exist or that consumers can subscribe.

**Fix:** Create topics explicitly (recommended for predictable behavior):

```
docker compose exec kafka kafka-topics --bootstrap-server kafka:29092 --
    create --topic order-events --partitions 3 --replication-factor 1
docker compose exec kafka kafka-topics --bootstrap-server kafka:29092 --
    create --topic inventory-events --partitions 3 --replication-factor 1
```

## 4.2 Phase 2: Docker Compose confusion (service missing / build context)

**Symptom:** Running `docker compose build order-service` produced:

```
no such service: order-service
```

**Diagnosis command:**

```
docker compose config
```

**Root cause:** The service wasn't present in the compose file that Docker Compose was reading (or the file path was wrong / edited file wasn't saved / service indentation name mismatch).

**Fix:** Ensure service block exists in `docker-compose.yml` and that folder paths match the repo structure, then rebuild:

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

## 4.3 Phase 3: "Site can't be reached" for the Order Service

**Symptom:** Browser couldn't reach the service, even though containers were running.


Figure 2: Browser error / site not reachable.

**Root cause:** The FastAPI process crashed at startup due to DB/Kafka not being ready or due to initialization errors.

**First debugging command:**

```
docker compose logs order-service
```

## 4.4 Phase 4: Postgres connection refused at startup

**Symptom:** In logs:

```
psycopg2.OperationalError: connection to server at "postgres" (172.18.0.2),
    port 5432 failed: Connection refused
Is the server running on that host and accepting TCP/IP connections?
```

**Meaning:** Postgres container existed, but DB server inside it wasn't ready yet when the service tried to connect.

**Fix options used during this project:**

- Restart the service after Postgres is up (`docker compose restart order-service`)
- Ensure SQLAlchemy session creation happens at request-time (not during module import) so it can recover

## 4.5 Phase 5: Kafka connection refused

**Symptom:** In logs:

```
%3|...|FAIL|rdkafka#producer-1| [thrd:kafka:29092/bootstrap]: kafka:29092/
   bootstrap: Connect ... failed: Connection refused
```

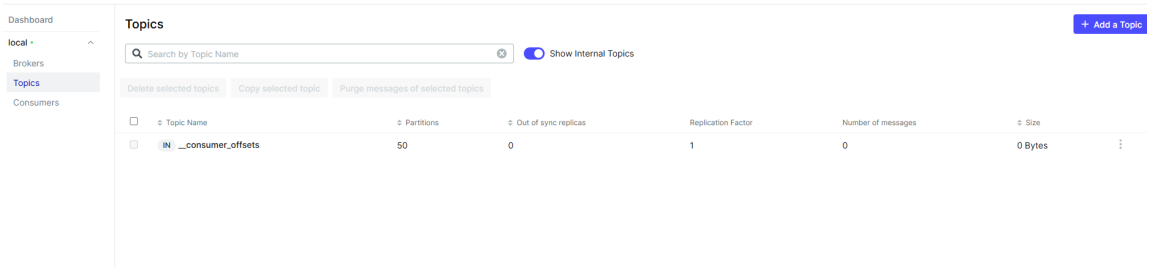


Figure 3: Kafka bootstrap connection refused.

**Root cause:** Kafka container starts quickly, but broker takes extra time to become ready.

**Fix:** Restart consumers once Kafka is ready and ensure internal networking uses `kafka:29092` (service-to-service), not localhost.

## 4.6 Phase 6: PowerShell curl confusion

**Symptom:** Using curl with `-H` failed because in PowerShell, `curl` is an alias for `Invoke-WebRequest`:

```
Invoke-WebRequest : Cannot bind parameter 'Headers'. Cannot convert the "
    Content-Type: application/json" value...
```

**Fix:** Use PowerShell-native `Invoke-RestMethod`:

```
Invoke-RestMethod `
  -Method POST `
  -Uri http://localhost:8000/orders `
  -ContentType "application/json" `
  -Body '{"item":"Laptop","quantity":1}'
```

## 4.7 Phase 7: SQLAlchemy session factory became None (SessionLocal bug)

**Symptom:**

```
TypeError: 'NoneType' object is not callable
```

**Root cause (typical):** importing `SessionLocal` incorrectly or overwriting it, causing `SessionLocal` to be `None` at runtime.

**Fix approach:**

- Ensure `SessionLocal = sessionmaker(...)` is defined once in `db.py`
- Import the module, not the variable (avoid capturing stale references)
- Create the session inside the request handler / worker loop

## 4.8 Phase 8: Missing topics (UNKNOWN_TOPIC_OR_PARTITION)

**Symptom:**

```
KafkaError{code=UNKNOWN_TOPIC_OR_PART,val=3,str="Subscribed topic not
    available: order-events: Broker: Unknown topic or partition"}
```

**Fix:** Manually create topics (`order-events` and `inventory-events`) using Kafka CLI inside the Kafka container (see Phase 1).

### 4.9 Phase 9: Notification service crash: missing import

**Symptom:**

```
NameError: name 'os' is not defined. Did you forget to import 'os'?
```

**Fix:** Add `import os` to the notification worker.

### 4.10 Phase 10: Git/GitHub publishing issues

**Problem 1:** push failed with missing `main` branch:

```
error: src refspec main does not match any
error: failed to push some refs...
```

**Fix:** Make the first commit, then rename branch:

```
git add .
git commit -m "Initial commit"
git branch -M main
git push -u origin main
```

**Problem 2:** Git identity not set:

```
Author identity unknown
fatal: unable to auto-detect email address (got 'ASAD@DESKTOP-...')
```

**Fix:**

```
git config --global user.name "Asad"
git config --global user.email "your-email@example.com"
```

### 4.11 Final Proof: Kafka UI shows flowing events

## 5 Canonical Command Reference (Copy/Paste)

### 5.1 Build and start

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

### 5.2 See running containers

```
docker compose ps
```

### 5.3 View logs

```
docker compose logs order-service
docker compose logs inventory-service
docker compose logs notification-service
```

### 5.4 Create topics

```
docker compose exec kafka kafka-topics --bootstrap-server kafka:29092 --
    create --topic order-events --partitions 3 --replication-factor 1
docker compose exec kafka kafka-topics --bootstrap-server kafka:29092 --
    create --topic inventory-events --partitions 3 --replication-factor 1
```

### 5.5 Test API (PowerShell)

```
Invoke-RestMethod `
  -Method POST `
  -Uri http://localhost:8000/orders `
  -ContentType "application/json" `
  -Body '{"item":"Phone","quantity":2}'
```

### 5.6 Check database

```
docker compose exec postgres psql -U app -d appdb
SELECT * FROM orders;
SELECT * FROM inventory;
SELECT * FROM notifications;
\q
```

### 5.7 Clean restart

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

## 6 Complete Source Code (All Files Under services/)

**Note:** The following code is included exactly as it exists in the project ZIP.

### 6.1 services/inventory-service/Dockerfile

Listing 1: services/inventory-service/Dockerfile

```
FROM python:3.12-slim

WORKDIR /app

COPY requirements.txt .
```

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

COPY app ./app

CMD ["python", "-m", "app.worker"]
```

## 6.2 services/inventory-service/app/db.py

Listing 2: services/inventory-service/app/db.py

```
import os
import time
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, DeclarativeBase
from sqlalchemy.exc import OperationalError

DATABASE_URL = os.environ["DATABASE_URL"]

engine = None
SessionLocal = None

class Base(DeclarativeBase):
    pass

def init_db(retries=10, delay=2):
    global engine, SessionLocal
    for i in range(retries):
        try:
            engine = create_engine(DATABASE_URL, pool_pre_ping=True)
            SessionLocal = sessionmaker(bind=engine)
            engine.connect().close()
            print("✅ Inventory DB connected")
            return
        except OperationalError:
            print(f"⏳ Inventory DB retry {i+1}/{retries}")
            time.sleep(delay)
    raise RuntimeError("❌ Inventory DB failed")

def get_session():
    if SessionLocal is None:
        raise RuntimeError("❌ Inventory DB not initialized")
    return SessionLocal()
```

## 6.3 services/inventory-service/app/kafka_client.py

Listing 3: services/inventory-service/app/kafka_client.py

```
import os, json
from confluent_kafka import Consumer, Producer

BOOTSTRAP = os.environ["KAFKA_BOOTSTRAP_SERVERS"]

def create_consumer():
    return Consumer({
        "bootstrap.servers": BOOTSTRAP,
```

```
        "group.id": "inventory-service",
        "auto.offset.reset": "earliest",  # ?? REQUIRED
        "enable.auto.commit": True
    })

def create_producer():
    return Producer({
        "bootstrap.servers": BOOTSTRAP
    })

def publish_inventory_event(producer, event):
    producer.produce("inventory-events", json.dumps(event).encode())
    producer.flush()
```

## 6.4 services/inventory-service/app/models.py

Listing 4: services/inventory-service/app/models.py

```
from sqlalchemy import Integer, String
from sqlalchemy.orm import Mapped, mapped_column
from .db import Base

class Inventory(Base):
    __tablename__ = "inventory"

    id: Mapped[int] = mapped_column(primary_key=True)
    order_id: Mapped[int] = mapped_column(Integer)
    item: Mapped[str] = mapped_column(String(200))
    quantity: Mapped[int] = mapped_column(Integer)
    status: Mapped[str] = mapped_column(String(50))
```

## 6.5 services/inventory-service/app/worker.py

Listing 5: services/inventory-service/app/worker.py

```
import json
from . import db
from .models import Inventory
from .kafka_client import create_consumer, create_producer,
   publish_inventory_event

def main():
    db.init_db()
    db.Base.metadata.create_all(bind=db.engine)

    consumer = create_consumer()
    producer = create_producer()

    consumer.subscribe(["order-events"])
    print("?? Inventory Service listening to order-events")

    while True:
        msg = consumer.poll(1.0)
        if msg is None:
            continue
```

```
        if msg.error():
            print("Kafka error:", msg.error())
            continue

        event = json.loads(msg.value().decode())
        print("➡ Received:", event)

        session = db.get_session()

        inv = Inventory(
            order_id=event["orderId"],
            item=event["item"],
            quantity=event["quantity"],   # ✅ FIXED
            status="RESERVED"
        )

        session.add(inv)
        session.commit()
        session.close()

        publish_inventory_event(producer, {
            "orderId": event["orderId"],
            "status": "RESERVED"
        })

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

## 6.6 services/inventory-service/requirements.txt

Listing 6: services/inventory-service/requirements.txt

```
SQLAlchemy
psycopg2-binary
confluent-kafka
```

## 6.7 services/notification-service/Dockerfile

Listing 7: services/notification-service/Dockerfile

```
FROM python:3.12-slim

WORKDIR /app

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

COPY app ./app

CMD ["python", "-m", "app.worker"]
```

## 6.8 services/notification-service/app/db.py

Listing 8: services/notification-service/app/db.py

```python
import os, time
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, DeclarativeBase
from sqlalchemy.exc import OperationalError

DATABASE_URL = os.environ["DATABASE_URL"]

engine = None
SessionLocal = None

class Base(DeclarativeBase):
    pass

def init_db(retries=10, delay=2):
    global engine, SessionLocal
    for i in range(retries):
        try:
            engine = create_engine(DATABASE_URL, pool_pre_ping=True)
            SessionLocal = sessionmaker(bind=engine)
            engine.connect().close()
            print("✅ Notification DB connected")
            return
        except OperationalError:
            print(f"⏳ Notification DB retry {i+1}/{retries}")
            time.sleep(delay)
    raise RuntimeError("❌ Notification DB failed")

def get_session():
    if SessionLocal is None:
        raise RuntimeError("❌ Notification DB not initialized")
    return SessionLocal()
```

## 6.9 services/notification-service/app/models.py

Listing 9: services/notification-service/app/models.py

```python
from sqlalchemy import Integer, String
from sqlalchemy.orm import Mapped, mapped_column
from .db import Base

class Notification(Base):
    __tablename__ = "notifications"

    id: Mapped[int] = mapped_column(primary_key=True)
    order_id: Mapped[int] = mapped_column(Integer)
    message: Mapped[str] = mapped_column(String(500))
```

## 6.10 services/notification-service/app/worker.py

Listing 10: services/notification-service/app/worker.py

```python
import os
import json
```

```
from confluent_kafka import Consumer
from . import db
from .models import Notification

def main():
    db.init_db()
    db.Base.metadata.create_all(bind=db.engine)

    consumer = Consumer({
        "bootstrap.servers": os.environ["KAFKA_BOOTSTRAP_SERVERS"],
        "group.id": "notification-service",
        "auto.offset.reset": "earliest"
    })

    consumer.subscribe(["inventory-events"])
    print("🔔 Notification Service listening to inventory-events")

    while True:
        msg = consumer.poll(1.0)
        if msg is None:
            continue
        if msg.error():
            print("Kafka error:", msg.error())
            continue

        event = json.loads(msg.value().decode())
        message = f"Order {event['orderId']} is {event['status']}"

        print("📢 Sending notification:", message)

        session = db.get_session()
        session.add(Notification(order_id=event["orderId"], message=message
            ))
        session.commit()
        session.close()

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

## 6.11 services/notification-service/requirements.txt

Listing 11: services/notification-service/requirements.txt

```
SQLAlchemy
psycopg2-binary
confluent-kafka
```

## 6.12 services/order-service/Dockerfile

Listing 12: services/order-service/Dockerfile

```
FROM python:3.12-slim

WORKDIR /app

```

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

COPY app ./app

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

## 6.13 services/order-service/app/db.py

Listing 13: services/order-service/app/db.py

```
import os
import time
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, DeclarativeBase
from sqlalchemy.exc import OperationalError

DATABASE_URL = os.environ["DATABASE_URL"]

engine = None
SessionLocal = None

class Base(DeclarativeBase):
    pass

def init_db(retries=10, delay=2):
    global engine, SessionLocal
    for i in range(retries):
        try:
            engine = create_engine(DATABASE_URL, pool_pre_ping=True)
            SessionLocal = sessionmaker(bind=engine)
            engine.connect().close()
            print("✅ Connected to Postgres")
            return
        except OperationalError:
            print(f"⏳ Postgres not ready, retrying ({i+1}/{retries})...")
            time.sleep(delay)

    raise RuntimeError("❌ Could not connect to Postgres")

def get_session():
    if SessionLocal is None:
        raise RuntimeError("❌ DB not initialized")
    return SessionLocal()
```

## 6.14 services/order-service/app/kafka_client.py

Listing 14: services/order-service/app/kafka_client.py

```
import os
import json
from confluent_kafka import Producer

producer = None

```

```
def get_producer():
    global producer
    if producer is None:
        producer = Producer({
            "bootstrap.servers": os.environ["KAFKA_BOOTSTRAP_SERVERS"]
        })
    return producer

def publish_order(order):
    event = {
        "orderId": order.id,
        "item": order.item,
        "quantity": order.quantity
    }

    p = get_producer()
    p.produce("order-events", json.dumps(event).encode("utf-8"))
    p.flush()
```

## 6.15 services/order-service/app/main.py

Listing 15: services/order-service/app/main.py

```
from fastapi import FastAPI, Depends
from pydantic import BaseModel
from sqlalchemy.orm import Session

from . import db
from .models import Order
from .kafka_client import publish_order

app = FastAPI(title="Order Service")

@app.on_event("startup")
def startup():
    db.init_db()
    db.Base.metadata.create_all(bind=db.engine)

class OrderCreate(BaseModel):
    item: str
    quantity: int

def get_db():
    session = db.get_session()
    try:
        yield session
    finally:
        session.close()

@app.post("/orders")
def create_order(req: OrderCreate, db_session: Session = Depends(get_db)):
    order = Order(item=req.item, quantity=req.quantity)
    db_session.add(order)
    db_session.commit()
    db_session.refresh(order)

```

```
    publish_order(order)

    return {
        "id": order.id,
        "item": order.item,
        "quantity": order.quantity
    }
```

### 6.16 services/order-service/app/models.py

Listing 16: services/order-service/app/models.py

```
from sqlalchemy import Integer, String
from sqlalchemy.orm import Mapped, mapped_column
from .db import Base

class Order(Base):
    __tablename__ = "orders"

    id: Mapped[int] = mapped_column(Integer, primary_key=True)
    item: Mapped[str] = mapped_column(String(200))
    quantity: Mapped[int] = mapped_column(Integer)
```

### 6.17 services/order-service/requirements.txt

Listing 17: services/order-service/requirements.txt

```
fastapi
uvicorn[standard]
SQLAlchemy
psycopg2-binary
confluent-kafka
pydantic
```

## 7 How to Upload and Compile on Overleaf

### 7.1 Upload steps

1. Create a new Overleaf project (*Blank Project*).
2. Upload this `main.tex` file.
3. Create a folder named `images` in Overleaf.
4. Upload the following files into `images/`:
   - 0886dbc2-3fc7-4325-a958-b97f472bab55.png
   - 09d137d6-f33e-4e8d-a642-3cc0cf3da3fb.png
   - 44ae6790-de48-40b3-9cb9-bead2595df00.png
   - 61272b96-339d-4b30-b3c7-1109857cdbd4.png

- 95a1b376-157a-49a7-90ac-098b6120b2a4.png
- ccd87be8-6acc-4482-807a-f894995d2745.png

5. In Overleaf: Menu → Settings → Compiler → select **XeLaTeX**.
6. Click **Recompile**.

## 7.2 If Overleaf shows "Unknown main document"

Ensure the main file is named `main.tex` or set it as main: Menu → Main document.

## 7.3 If Overleaf cannot render PDF

Overleaf sometimes displays a browser-side PDF rendering error due to blocked domains. Try:

- Disable strict adblock/privacy extensions for Overleaf
- Try another browser
- Download the PDF instead of in-browser preview

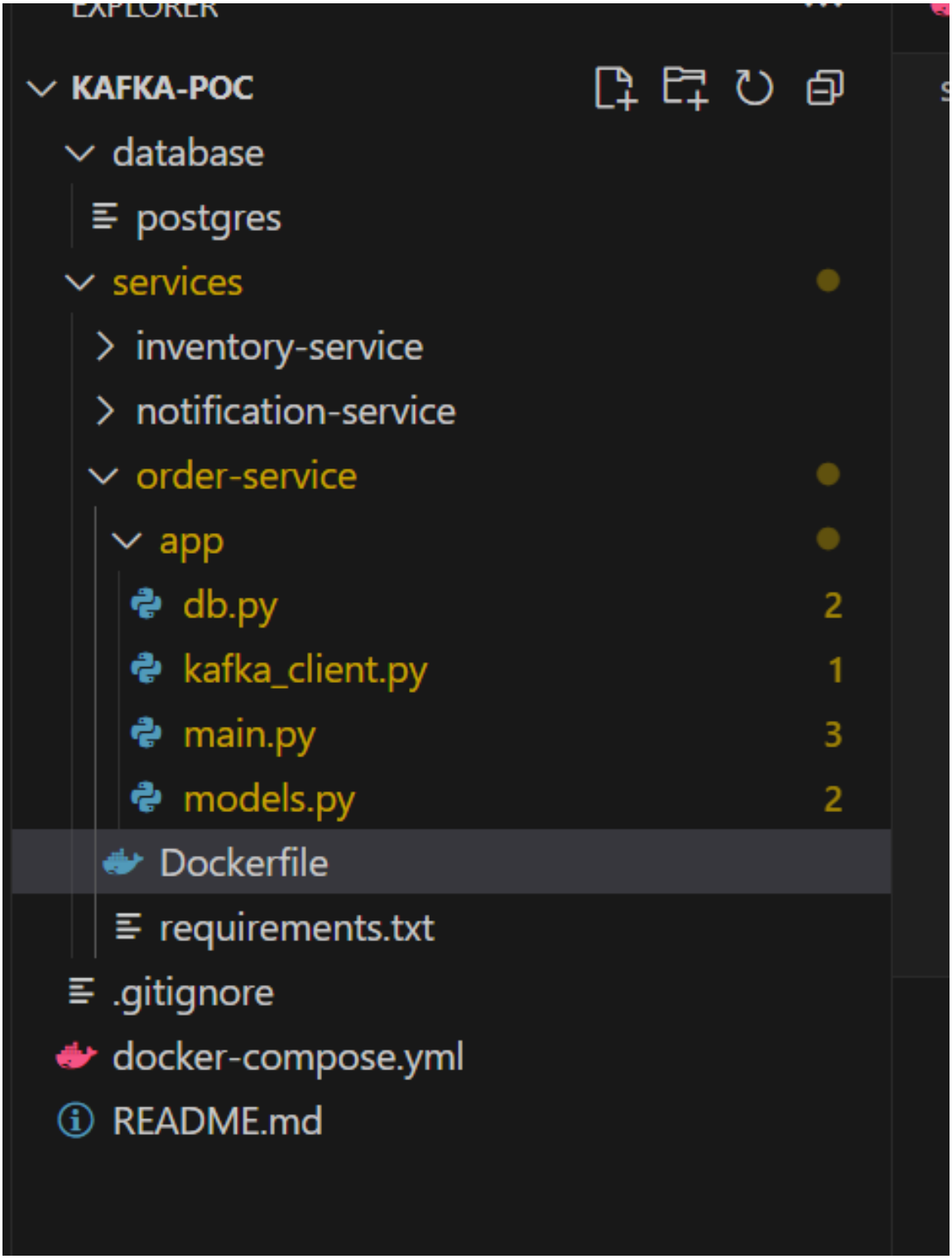


Figure 4: Terminal log showing service/runtime errors during initialization.

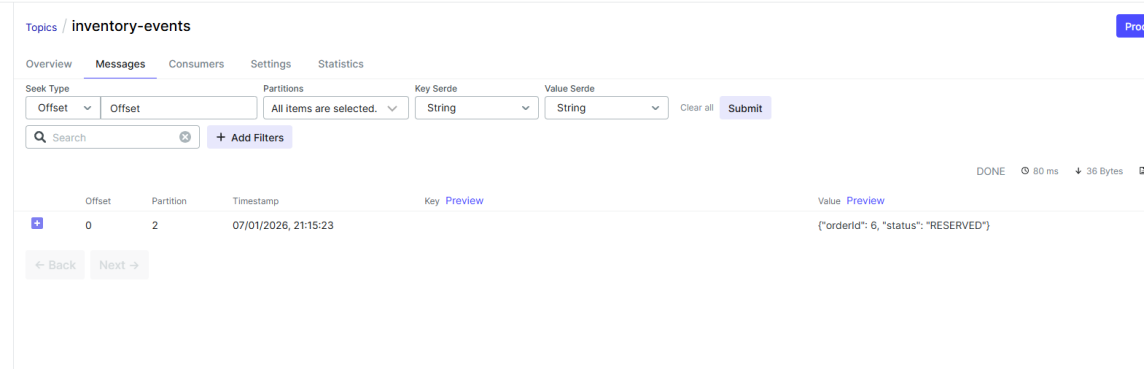


Figure 5: Kafka UI view showing application events after fixes.