FastAPI Message Broker

A Minimal In-Memory Message Broker System

Similar to Apache Kafka

Built with Python & FastAPI

What We Are Building

System Components & Architecture

Three Core Components

- ✓ **Broker** - FastAPI server that stores messages per topic
- ✓ **Producer** - Python script that publishes messages via HTTP
- ✓ **Consumer** - Python script that polls and reads messages

Key Constraints

- Data exists only in RAM (in-memory)
- One message list per topic
- No partitions or replication
- Each consumer has independent offset tracking

Project Structure

Organised for Clarity & Maintainability

```
mini-kafka/ ├─ broker/ │ └─ main.py # FastAPI app (the broker) ├─ producer/ │ └─ main.py # Producer script ├─ consumer/ │ └─
main.py # Consumer script ├─ shared/ │ └─ schemas.py # Pydantic models shared by all apps ├─ requirements.txt └─ README.md
```

Why This Split?

- broker/ - Web server with REST endpoints
- producer/consumer/ - Plain Python HTTP clients
- shared/ - Avoids duplicating request/response models

Data Models (Pydantic)

Type-Safe Request & Response Schemas

Model	Purpose	Fields
TopicRegistration	Ensure topic exists	{ topic: str }
ProducerRegistration	Register a producer	{ topic: str, producer_id?: str }
ConsumerRegistration	Register a consumer	{ topic: str, consumer_id?: str }
PublishRequest	Publish a message	{ topic: str, value: str, key?: str }
ConsumeRequest	Request next message	{ consumer_id: str }
Message	Message response	{ topic: str, offset: int, value: str, key?: str }

API Endpoints

RESTful Interface with Auto-Generated Swagger Docs

Method	Endpoint	Purpose
GET	/	Welcome message
GET	/health	Health check
POST	/topics/register	Create topic array if missing
POST	/producers/register	Acknowledge producer (logging/demo)
POST	/consumers/register	Create consumer, set offset to 0
POST	/produce	Append message, assign offset
POST	/consume	Return next message, increment offset
GET	/stats	Return counts per topic and consumer offsets

Consumer Workflow

Polling & Reading Messages

```
Consumer                   Broker
   |  POST /topics/register  |
   |------------------------>|  ensure topic exists
   |                         |
   |  POST /consumers/register (topic)
   |------------------------>|  {consumer_id, offset=0}
   |                         |
Repeat (infinite loop):
   |  POST /consume {consumer_id}
   |------------------------>|
   |  <----------------------|  200 {message} or
   |                         |  204 (no content)
```

Response Handling

- **200 OK** - Message available, print and continue
- **204 No Content** - No new messages, sleep for poll interval

Understanding Offsets

Independent Tracking Per Consumer

Example: Topic "demo" with messages [0, 1, 2]

- Consumer **alice** starts with offset = 0
- First /consume → returns message at index 0, offset becomes 1
- Next /consume → returns index 1, offset becomes 2
- If no message at index 2, broker returns 204

Multiple Consumers

Each consumer's offset is **independent**. A second consumer **bob** also starts at offset 0 and will receive all messages from the beginning.

Key Point: Offsets are per-consumer, enabling parallel independent consumption

Key Technical Features

What Makes This System Robust

Architecture Benefits

- ✓ Async operations with FastAPI
- ✓ Thread-safe with per-topic locks
- ✓ Type-safe with Pydantic models
- ✓ Auto-generated API documentation
- ✓ Modular component separation

Message Guarantees

- ✓ Message ordering within topics
- ✓ Independent consumer offsets
- ✓ Configurable polling intervals
- ✓ Built-in monitoring via /stats
- ✓ HTTP-based, language-agnostic

Common Pitfalls & Solutions

Troubleshooting Guide

Issue	Cause	Solution
204 No Content	No new messages available	Not an error - keep polling
404 Topic Not Found	Topic not registered	Call /topics/register first
Data Lost on Restart	In-memory storage by design	Add disk persistence if needed
Consumer Conflicts	Duplicate consumer IDs	Use unique --consumer-id per consumer
Low Throughput	Single-process limitations	Add partitions, batching, or workers

How to Extend the System

Future Enhancement Opportunities

Performance & Scale

- **Partitions** - Multiple shards per topic
- **Batching** - Produce/consume multiple messages
- **Backpressure** - Max topic size limits
- **Long-polling** - Reduce polling overhead

Enterprise Features

- **Durability** - SQLite or file persistence
- **Consumer Groups** - Load balancing
- **Auth & ACLs** - Security controls
- **Offset Commits** - At-least-once semantics

Summary

What We've Built

Core Achievement

A **lightweight, maintainable message broker** system with clear separation of concerns, built on modern Python async principles.

Key Takeaways

✓ Simple yet functional message broker architecture

✓ Async FastAPI for high-performance HTTP endpoints

✓ Independent offset tracking for multiple consumers

✓ Type-safe with Pydantic, developer-friendly with Swagger

✓ Foundation for learning distributed systems concepts

Perfect for: Learning, prototyping, and understanding message broker fundamentals

Thank You

Questions?

FastAPI Message Broker

Built with Python, FastAPI & Pydantic