Kafka API Wrapper – Developer Documentation

1. Project Overview

Kafka API Wrapper is a modular **Spring Boot application** designed to facilitate seamless message publishing and consumption using **Apache Kafka**.
It includes:

- AES-based encryption utilities (`AESUtil`)
- JWT-based authentication (`JwtUtil`)
- Docker-based deployment for easy setup and testing

This project enables microservices or standalone applications to send and receive Kafka messages through secure REST APIs.

---

2. Technology Stack

- Java 17 / Spring Boot
- Apache Kafka
- Docker & Docker Compose
- Maven Build Tool
- JSON Web Token (JWT) Authentication
- Postman for API testing
- `gen-jwt.js` (Node.js script) for generating JWT tokens

---

3. Project Structure (Simplified)

```
kafka-api-wrapp/
├── pom.xml                     # Maven configuration
├── docker-compose.yml          # Docker setup for Kafka and Zookeeper
├── gen-jwt.js                  # JWT Token generation script
(Node.js)
├── src/
│   ├── main/java/com/example/kafkaap/wrapper/
│   │   ├── controller/TransactionController.java
│   │   ├── kafka/KafkaProducerService.java
│   │   ├── kafka/KafkaConsumerService.java
│   │   ├── util/AESUtil.java
│   │   ├── util/JwtUtil.java
│   │   ├── util/ChecksumUtil.java
│   │   ├── config/KafkaConfig.java
│   │   ├── config/WebConfig.java
│   └── test/java/...           # Unit tests
```

---

4. Setup and Execution Steps

Step 1: Start Docker Services

Ensure Docker is installed and running.
From the project root, execute:

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

This will start **Kafka** and **Zookeeper** containers.

Step 2: Build and Run the Spring Boot Application

Use Maven to build and run the service:

```
mvn clean install
mvn spring-boot:run
```

The API will be available at:
👉 **http://localhost:8086**

Step 3: Generate JWT Token (Authorization)

Use the provided Node.js script `gen-jwt.js` to generate a valid JWT token:

```
node gen-jwt.js
```

You'll get a token like:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Use it in Postman under **Headers**:

```
Authorization: Bearer <your_token>
```

---

5. Testing via Postman

Endpoint: Send Message to Kafka

POST → `http://localhost:8086/api/tx/send`

Headers:

```
Content-Type: application/json
Authorization: Bearer <token>
```

Body:

```
{
  "topic": "sample-topic",
  "message": "Hello Kafka"
}
```

Expected Response:

```
{
  "status": "Message sent successfully",
  "topic": "sample-topic"
}
```

Endpoint: Receive Messages from Kafka

If your application also includes a consumer controller, you can use:

```
GET http://localhost:8086/api/tx/send
```

Response:

```
{
  "messages": ["Hello Kafka", "Another message"]
}
```

---

6. JWT Tokenization with gen-jwt.js

This file (`gen-jwt.js`) is a lightweight Node.js script to generate JWT tokens.

Sample:

```
const jwt = require('jsonwebtoken');
const token = jwt.sign({ sub: 'admin' }, 'secret-key', { expiresIn:
'1h' });
console.log(token);
```

Run with:

```
node gen-jwt.js
```

---

7. Reusability (as Library JAR)

The Kafka API Wrapper can be repackaged as a reusable JAR for integration in other projects.

Build the JAR:

```
mvn clean package
```

Generated File:

```
target/kafka-api-wrapper-1.0.0.jar
```

You can now include it in other projects by adding it to the classpath or as a Maven dependency.

---

8. Developer Notes & Best Practices

- Ensure Docker containers are running before starting Spring Boot.
- Verify that **Kafka broker configs** in Docker and `application.yml` match.
- Use `gen-jwt.js` for generating valid tokens during testing.
- Extend `JwtUtil.java` if you want to support token creation as well as validation.
- `AESUtil` and `ChecksumUtil` can be reused in other services for data encryption and integrity verification.
- Keep Postman collections updated for consistent API testing.

---

9. Common Troubleshooting

Issue	Cause	Solution
`Kafka broker not available`	Docker not running	Run `docker-compose up -d`
`JWT invalid`	Token expired	Regenerate with `node gen-jwt.js`

`Connection refused: localhost:9092`	Kafka container not ready	Wait 10-15 seconds after Docker start
`NoClassDefFoundError` when reusing JAR	Missing dependencies	Include `jjwt-api`, `jjwt-impl`, and `jjwt-jackson` jars

---

10. Contribution and Contact

For future contributions:

- Follow Git branching standards (`feature/`, `bugfix/`, `release/`)
- Include unit tests for new code
- Document API changes clearly

Maintainer: Varun Baral
Environment Used: macOS, Java 17, Docker Desktop, Maven 3.9+, Postman 11+

⚙️ Using the kafka-api-wrapper JAR in Another Project (Example)

When you build your project with Spring Boot, it generates a **fat JAR** (contains dependencies under `BOOT-INF/classes`).
To reuse it in other Java projects or simple test programs, you'll first convert it into a **flat JAR** and then include it with required dependencies.

Step 1: Navigate to the JAR Directory

```
cd lib
```

Step 2: Extract the Fat JAR

```
mkdir extracted
cd extracted
```

```
jar xf ../kafka-api-wrapper-1.0.0.jar
```

After extraction, the structure looks like:

```
extracted/
 └── BOOT-INF/
     └── classes/
         └── com/example/kafkaap/wrapper/util/...
```

Step 3: Repackage Classes into a Flat JAR

```
cd BOOT-INF/classes
jar cf ../../../kafka-api-wrapper-flat.jar com
cd ../../../
```

✅ You now have a clean, usable `lib/kafka-api-wrapper-flat.jar`

Step 4: Download Required JWT Dependencies

```
curl -O
https://repo1.maven.org/maven2/io/jsonwebtoken/jjwt-api/0.11.5/jjwt-ap
i-0.11.5.jar
curl -O
https://repo1.maven.org/maven2/io/jsonwebtoken/jjwt-impl/0.11.5/jjwt-i
mpl-0.11.5.jar
curl -O
https://repo1.maven.org/maven2/io/jsonwebtoken/jjwt-jackson/0.11.5/jjw
t-jackson-0.11.5.jar
```

Place them all inside your **`lib/`** folder.

Step 5: Create a Simple Test Project

Folder Structure:

```
jar-test/
 ├── lib/
 │   ├── kafka-api-wrapper-flat.jar
 │   ├── jjwt-api-0.11.5.jar
 │   ├── jjwt-impl-0.11.5.jar
 │   └── jjwt-jackson-0.11.5.jar
 ├── src/
 │   └── TestJarMain.java
 └── out/
```

Step 6: Compile and Run

```
javac -cp "lib/*" -d out src/TestJarMain.java
java -cp "lib/*:out" TestJarMain
```

✅ If configured correctly, you'll see:

```
=== Testing kafka-api-wrapper.jar ===
Encrypted: <ciphertext>
Decrypted: <plaintext>
JWT Valid: true
=== JAR Test Completed ===
```