

README for NATS Subscriber:

3-Layer Java Application with PostgreSQL Persistence

Project Description

This is a simple Java application that subscribes to messages published on a NATS (NATS.io) subject and stores them in a PostgreSQL database. It uses a clean three-layer architecture (API, Service, Data), and can be run either locally or via Docker Compose. The project includes a schema setup script for the database and testable service/data logic.

Requirements

Before running the project, make sure the following tools are installed on your machine:

1. Java 21 – Required to compile and run the application.
Download: <https://adoptium.net>
2. Maven – Used to build the project and generate the executable JAR file.
Download: <https://maven.apache.org/download.cgi>
3. PostgreSQL – The database that stores incoming messages.
Download: <https://www.postgresql.org/download/>
4. NATS Server – A lightweight messaging system (pub/sub) that this app connects to.
Download: <https://docs.nats.io/running-a-nats-service/introduction>
5. NATS CLI – A command-line tool to publish test messages to the NATS server.
Download: <https://github.com/nats-io/natscli/releases>
6. Docker Desktop – Used if you prefer to run everything using Docker Compose.
Download: <https://www.docker.com/products/docker-desktop>

3-Layer Architecture

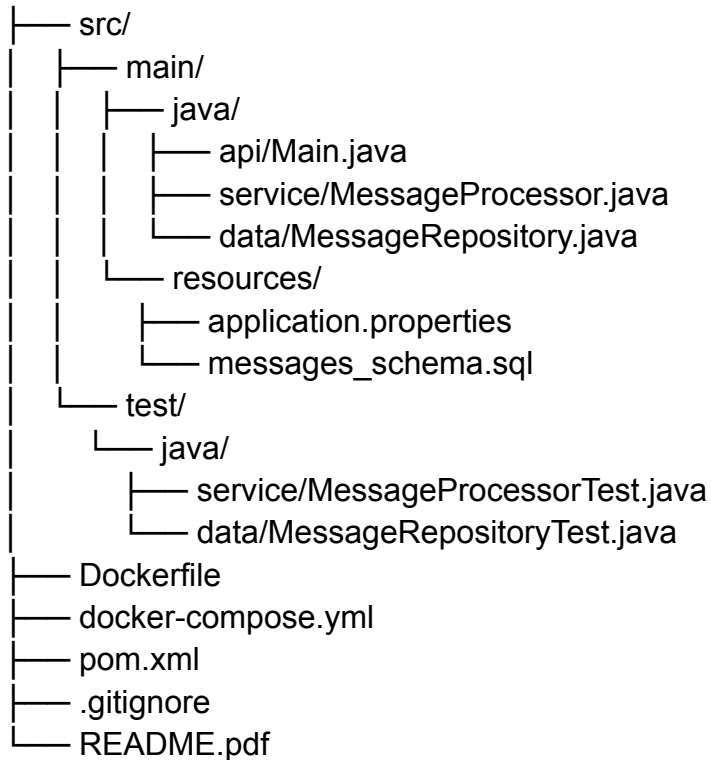
The project follows a standard three-layered design:

- API (api/Main.java) - Starts the app, connects to NATS, listens for messages
- Service (service/MessageProcessor.java) - Validates and processes incoming messages
- Data (data/MessageRepository.java) - Handles database connection and message storage

Each layer is modular and follows separation of concerns principles.

Project Structure

NatsSubscriber/



Step-by-Step Setup Instructions

1. PostgreSQL Setup

Option A: Local Setup

- Run PostgreSQL
- Create a database named: *CREATE DATABASE nats_messages;*
- Run the schema setup file `src/main/resources/messages_schema.sql`.
- **Important:** Update `src/main/resources/application.properties` to match your own PostgreSQL credentials, for example:

```
db.url=jdbc:postgresql://localhost:5432/nats_messages
db.user=your_pg_username
db.password=your_pg_password
```

Each system may use a different PostgreSQL username and password depending on your installation and setup. Be sure to use your actual credentials.

Option B: Docker

- PostgreSQL is pre-configured via Docker Compose with:
- Username: postgres
- Password: 152535
- Database: nats_messages

No manual credential configuration is required when running the Docker setup.

2. NATS Server Setup

Option A: Local

- Run the `nats-server.exe` file in a terminal: `nats-server`

Option B: Docker

NATS will start automatically with Docker Compose.

3. Run the Application

Option A: Local (IntelliJ)

- Ensure PostgreSQL and NATS are running
- Open the project in IntelliJ
- Run `api/Main.java`
- You should see:
Connecting to NATS...
Subscribed to 'updates'. Waiting for messages...

Option B: Docker Compose

- Build the project: `mvn clean package`
- Start all services:
docker compose build
docker compose up
- Logs will confirm the app is running and ready to receive messages.

4. Send a Test Message

- From NATS CLI: `nats pub updates "Message from CLI"`
- Expected application output:
Received message: Message from CLI
Processing and saving message: Message from CLI
- You can then verify it in the database:
*SELECT * FROM messages ORDER BY id DESC;*

Running Tests

- To run the unit tests: `mvn test`

Clone & Run from GitHub

- To clone this repository:
git clone https://github.com/anzheltorosyan/NatsSubscriber.git

cd NatsSubscriber

- To open in IntelliJ:
 - File → Open → Select the project folder
 - Let IntelliJ import the Maven project (pom.xml)
 - Run Main.java or use the Maven panel

Docker Compose Overview

Docker Compose sets up:

- PostgreSQL with schema
- NATS server
- Java app with environment-based configuration

To start: *docker compose up*

To stop: *docker compose down*

.gitignore

Recommended entries:

```
/target/      # Compiled files from Maven (not needed in version control)
/.idea/       # IntelliJ settings folder
/*.*.iml      # IntelliJ project module files
*.log         # Any log files
.env          # If you use environment variables locally
```

Author

Student: Anzhel Torosyan

Course: Software Engineering (Spring 2025)

Instructor: Vahe Momjyan

University: American University of Armenia