**Київський політехнічний інститут імені Ігоря Сікорського**

**Фізико-технічний інститут**

**Проектування розподілених систем**

**Проект**

Replicated log task

**Виконали:**

Студенти групи ФБ-42мп

Алькова Аліна, Легойда Юлія, Осіпчук Антон

**Iteration 0.**

Choose a desirable language for implementation and try to implement (or find the implementation) a simple *Echo Client-Server* application.

Для цієї частини ми реалізували клієнт-серверну програму мовою Python, використовуючи бібліотеку Flask для серверної частини та requests для клієнтської частини. Програма складається з клієнта і сервера, які взаємодіють через HTTP-запити

Серверна частина (server.py):

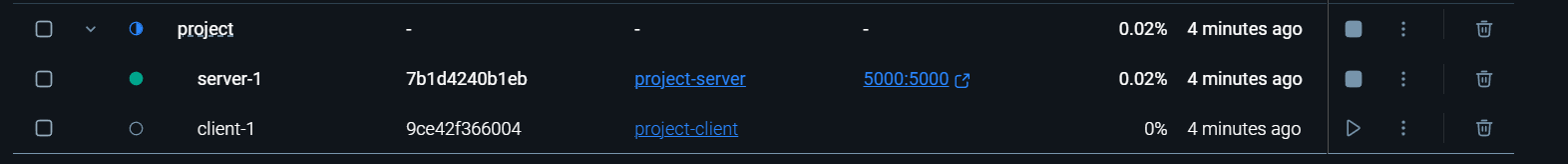
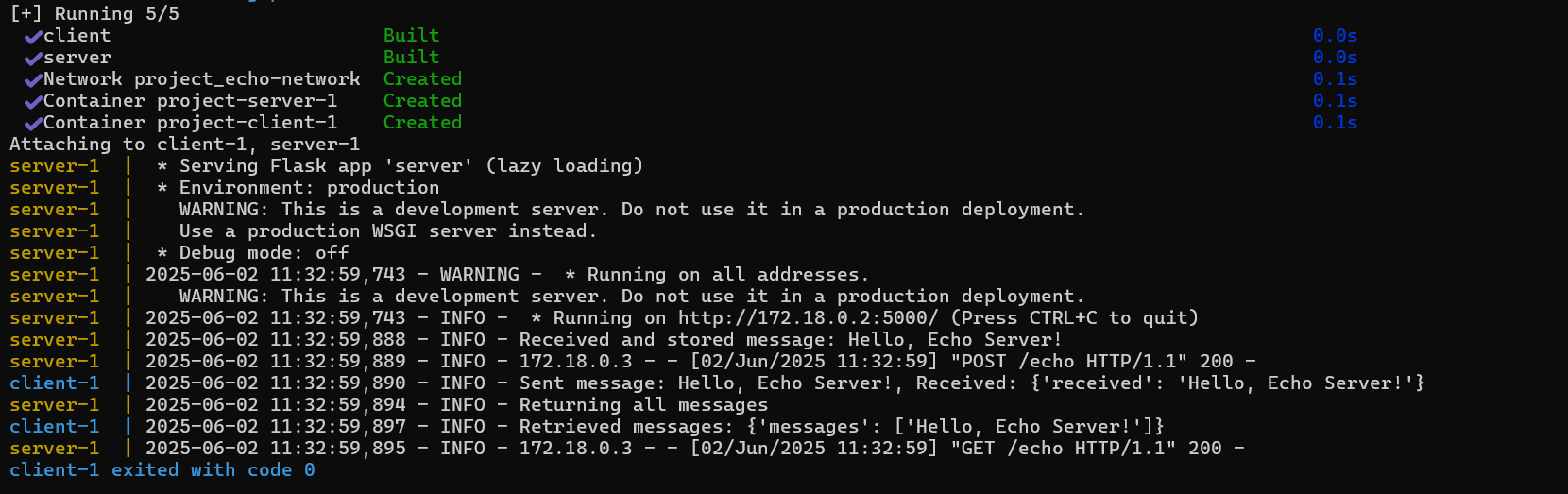
from flask import Flask, request, jsonify  
import logging  
  
app = Flask(\_\_name\_\_)  
  
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')  
logger = logging.getLogger(\_\_name\_\_)  
  
messages = []  
  
@app.route('/echo', methods=['POST'])  
def echo\_post():  
 data = request.get\_json()  
 if not data or 'message' not in data:  
 logger.error("Invalid request: 'message' field is required")  
 return jsonify({"error": "Message is required"}), 400  
  
 message = data['message']  
 messages.append(message)  
 logger.info(f"Received and stored message: {message}")  
 return jsonify({"received": message}), 200  
  
@app.route('/echo', methods=['GET'])  
def echo\_get():  
 logger.info("Returning all messages")  
 return jsonify({"messages": messages}), 200  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(host='0.0.0.0', port=5000)

* Сервер приймає HTTP-запити на ендпоінт /echo з методами POST і GET.
* **POST /echo**: Отримує JSON-об'єкт із полем message, зберігає повідомлення в список messages і повертає отримане повідомлення у відповіді з кодом 200. Якщо поле message відсутнє, повертається помилка 400.
* **GET /echo**: Повертає всі збережені повідомлення у вигляді JSON.
* Реалізовано логування за допомогою модуля logging для відстеження отриманих повідомлень та помилок.

Клієнтська частина (client.py):

import requests  
import logging  
  
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')  
logger = logging.getLogger(\_\_name\_\_)  
  
def send\_message(message):  
 url = 'http://server:5000/echo'  
 data = {'message': message}  
 try:  
 response = requests.post(url, json=data)  
 response.raise\_for\_status()  
 logger.info(f"Sent message: {message}, Received: {response.json()}")  
 except requests.RequestException as e:  
 logger.error(f"Failed to send message: {e}")  
  
def get\_messages():  
 url = 'http://server:5000/echo'  
 try:  
 response = requests.get(url)  
 response.raise\_for\_status()  
 logger.info(f"Retrieved messages: {response.json()}")  
 except requests.RequestException as e:  
 logger.error(f"Failed to retrieve messages: {e}")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 send\_message("Hello, Echo Server!")  
 get\_messages()

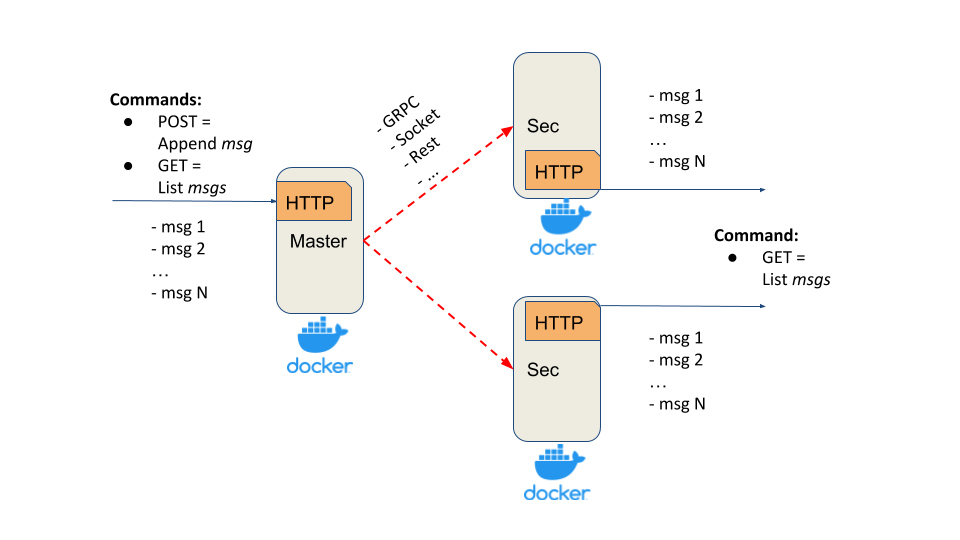
* Функція send\_message: Надсилає повідомлення на сервер через POST-запит до /echo.
* Функція get\_messages: Отримує всі повідомлення з сервера через GET-запит до /echo.
* Обробка помилок: Використовується try-except для обробки виключень, пов'язаних із мережевими запитами.
* **Логування**: Реалізовано логування для відстеження надісланих і отриманих даних, а також помилок.
* Клієнт автоматично надсилає тестове повідомлення "Hello, Echo Server!" і викликає функцію для отримання повідомлень.

Перевірка запуска:

**Iteration 1.**

* **5 points**

The Replicated Log should have the following deployment architecture: one ***Master*** and any number of ***Secondaries***.



***Master*** should expose a simple HTTP server (or alternative service with a similar API) with:

* *POST method* - appends a message into the in-memory list
* *GET method* - returns all messages from the in-memory list

***Secondary*** should expose a simple  HTTP server(or alternative service with a similar API)  with:

* *GET method* - returns all replicated messages from the in-memory list

Properties and assumptions:

* after each POST request, the message should be replicated on every *Secondary* server
* *Master* should ensure that *Secondaries* have received a message via *ACK*
* *Master’s POST request* should be finished only after receiving *ACKs* from all *Secondaries* (blocking replication approach)
* to test that the replication is blocking, introduce the delay/sleep on the *Secondary*
* at this stage assume that the communication channel is a perfect link (no failures and messages lost)
* any RPC framework can be used for *Master-Secondary* communication (Sockets, language-specific RPC, HTTP, Rest, gRPC, …)
* your implementation should support logging
* *Master* and *Secondaries* should run in Docker

У цій частині ми налаштували один майстер і два вторинних сервера (secondary1, secondary2) у docker-compose.yml. Архітектура підтримує можливість додавання нових вторинних серверів шляхом зміни конфігурації.

**Майстер (master.py):**

from flask import Flask, request, jsonify  
import requests  
import logging  
import time  
  
app = Flask(\_\_name\_\_)  
  
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')  
logger = logging.getLogger(\_\_name\_\_)  
  
messages = []  
  
SECONDARIES = ['http://secondary1:5001', 'http://secondary2:5001']  
  
@app.route('/messages', methods=['POST'])  
def post\_message():  
 data = request.get\_json()  
 if not data or 'message' not in data:  
 logger.error("Invalid request: 'message' field is required")  
 return jsonify({"error": "Message is required"}), 400  
  
 message = data['message']  
 messages.append(message)  
 logger.info(f"Stored message: {message}")  
  
 acks = []  
 for secondary in SECONDARIES:  
 try:  
 start\_time = time.time()  
 response = requests.post(f"{secondary}/replicate", json={'message': message}, timeout=10)  
 response.raise\_for\_status()  
 acks.append(True)  
 logger.info(f"Received ACK from {secondary} in {time.time() - start\_time:.2f}s")  
 except requests.RequestException as e:  
 logger.error(f"Failed to replicate to {secondary}: {e}")  
 return jsonify({"error": f"Failed to replicate to {secondary}"}), 500  
  
 if all(acks):  
 logger.info("All Secondaries acknowledged")  
 return jsonify({"status": "Message replicated"}), 200  
 else:  
 logger.error("Not all Secondaries acknowledged")  
 return jsonify({"error": "Replication failed"}), 500  
  
@app.route('/messages', methods=['GET'])  
def get\_messages():  
 logger.info("Returning all messages")  
 return jsonify({"messages": messages}), 200  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(host='0.0.0.0', port=5000)

* Реалізовано HTTP-сервер за допомогою Flask, який працює на порту 5000.
* **Ендпоінти**:
* **POST /messages**: Приймає JSON-запит із полем message, зберігає повідомлення у локальний список messages і відправляє його на всі вторинні сервери (Secondaries) через POST-запит до їхнього ендпоінта /replicate. Повертає відповідь лише після отримання підтверджень (ACK) від усіх вторинних серверів.
* **GET /messages**: Повертає всі збережені повідомлення у вигляді JSON.
* Використовується модуль logging для запису інформації про збережені повідомлення, успішні реплікації та помилки.
* **Реплікація**: Майстер відправляє повідомлення на два вторинні сервери (secondary1:5001, secondary2:5001) і чекає підтвердження (ACK) від кожного. Якщо хоча б один сервер не підтверджує, повертається помилка 500.

**Вторинні сервери (secondary.py):**

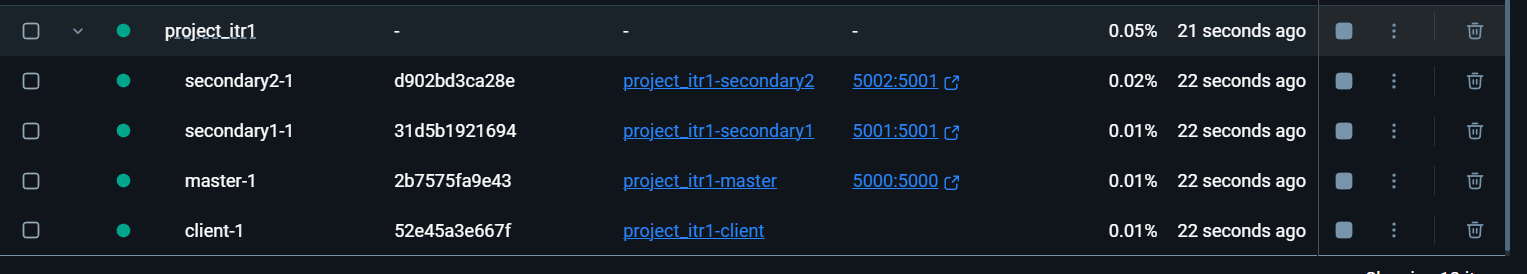
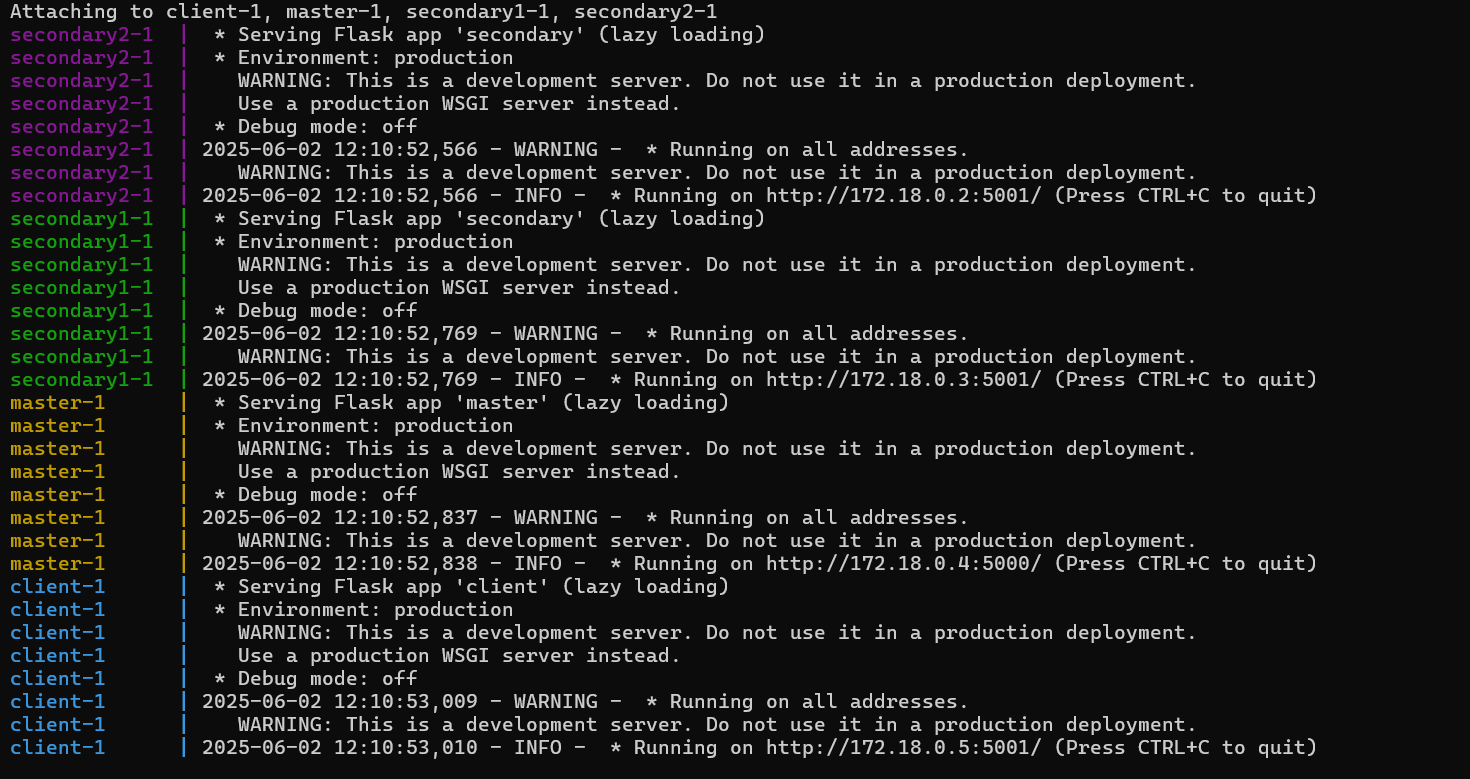
from flask import Flask, request, jsonify  
import logging  
import time  
  
app = Flask(\_\_name\_\_)  
  
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')  
logger = logging.getLogger(\_\_name\_\_)  
  
messages = []  
  
@app.route('/replicate', methods=['POST'])  
def replicate\_message():  
 data = request.get\_json()  
 if not data or 'message' not in data:  
 logger.error("Invalid request: 'message' field is required")  
 return jsonify({"error": "Message is required"}), 400  
  
 message = data['message']  
 time.sleep(2)  
 messages.append(message)  
 logger.info(f"Replicated message: {message}")  
 return jsonify({"status": "ACK"}), 200  
  
@app.route('/messages', methods=['GET'])  
def get\_messages():  
 logger.info("Returning all messages")  
 return jsonify({"messages": messages}), 200  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(host='0.0.0.0', port=5001)

* Реалізовано HTTP-сервер за допомогою Flask, який працює на порту 5001 (для secondary1 і secondary2, хоча для другого порт мапиться як 5002 зовні через Docker Compose).
* **Ендпоінти**:
* **POST /replicate**: Приймає повідомлення від майстра, додає затримку в 2 секунди (time.sleep(2)), зберігає повідомлення у локальний список messages і повертає підтвердження ({"status": "ACK"}).
* **GET /messages**: Повертає всі збережені повідомлення.
* Логуються отримані повідомлення та запити.

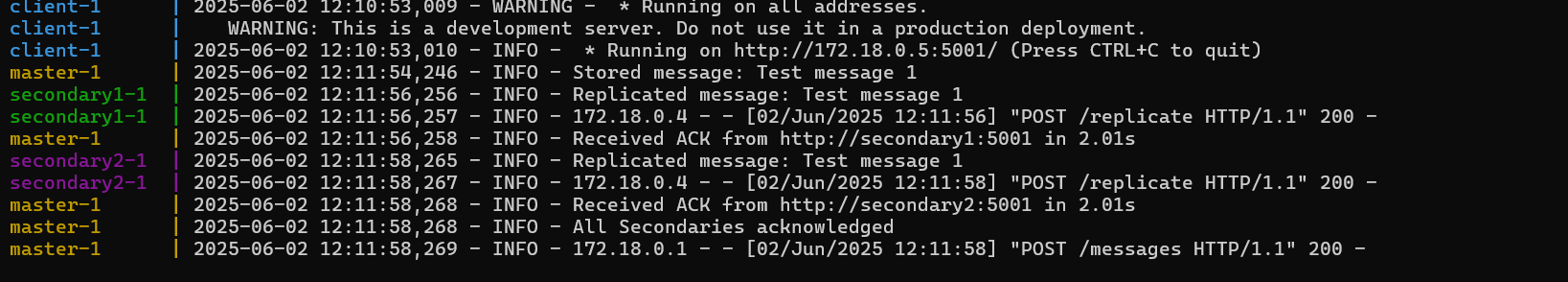
**Клієнт (client.py):**

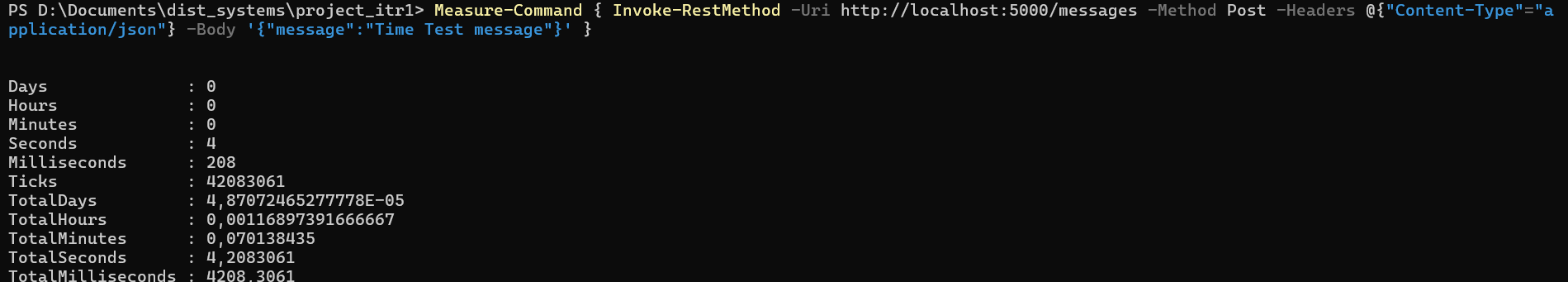
Незважаючи на назву "client", файл фактично дублює код вторинного сервера (secondary.py) і працює як ще один HTTP-сервер на порту 5001. Він має ті ж ендпоінти (/replicate і /messages) і ту ж логіку, що й вторинні сервери. Ми ніяк не використовуємо його повноцінно, ми залишили client.py як резервну копію вторинного сервера. Усі подальші тести ми проводимо вручну але можна було б в цьому клієнті реалізувати автоматичне надсилання потрібних запитів.

**Тестування роботи:**

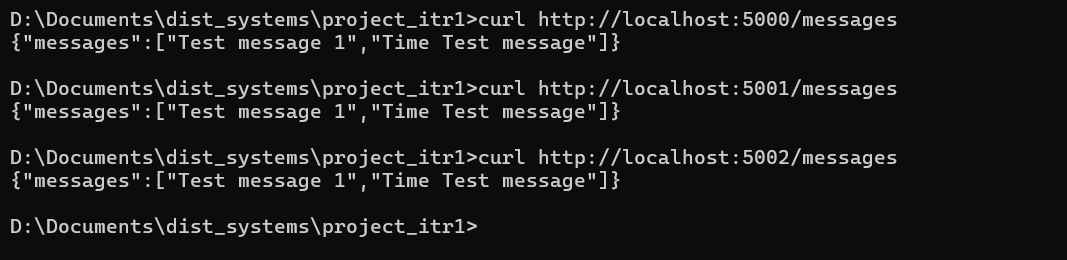
****

**Тестування POST-запиту до Master**:

Надсилаємо POST-запит до Master (http://localhost:5000/messages) із повідомленням, щоб перевірити, чи воно зберігається та реплікується на Secondary-сервери.

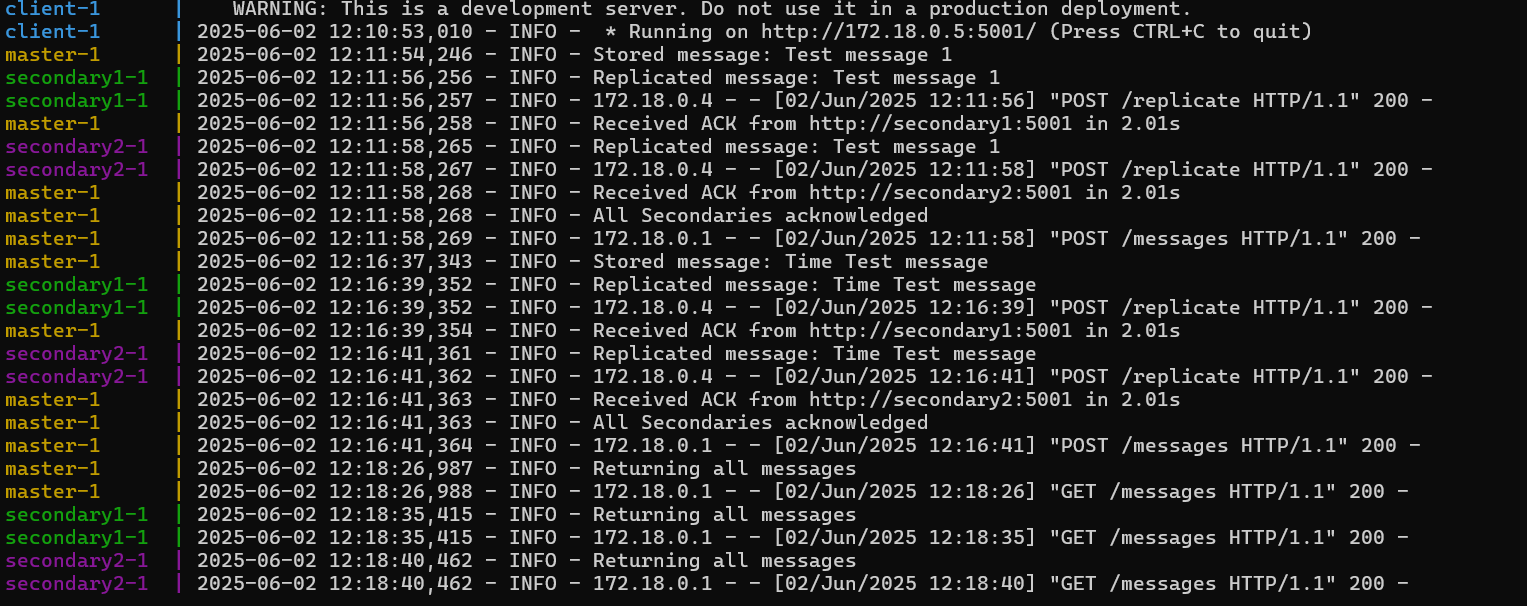
**Перевірка блокуючої реплікації**:

Вимірювання часу виконання POST-запиту підтверджує, що Master чекає на ACK від усіх Secondary, і затримка (2 секунди на кожен Secondary) впливає на загальний час. Це демонструє сильну консистентність: повідомлення не вважається збереженим, доки всі сервери його не підтвердять.

**Перевірка GET-запитів**:

Запити до Master і Secondary перевіряють, що повідомлення коректно зберігаються і синхронізуються між усіма серверами. Це підтверджує, що реплікація працює правильно.

**Перевірка логів**:

 Логи містять інформацію про збереження повідомлень, реплікацію та ACK.