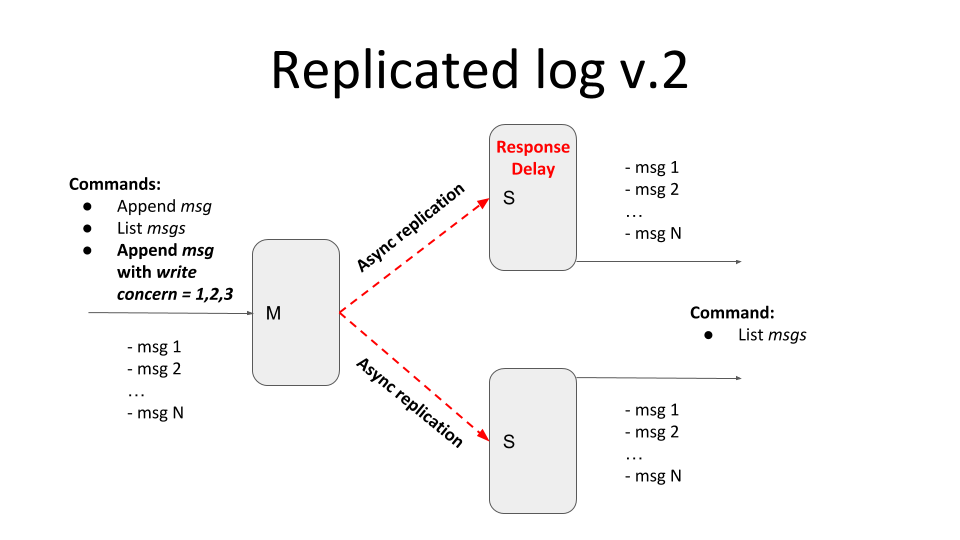
**Iteration 2.**

* **5 points**

In the previous iteration, the replication was blocking for all secondaries, i.e. to return a response to the client we should receive acknowledgements (ACK) from all secondaries.



Current iteration should provide tunable semi-synchronicity for replication, by defining *write concern* parameters.

* client POST request in addition to the message should also contain *write concern* parameter *w=1,2,3,..,n*
* *w* value specifies how many ACKs the master should receive from secondaries before responding to the client

*w = 1* - only from master

*w = 2* - from master and one secondary

*w = 3* - from master and two secondaries

Please emulate the replica’s inconsistency (and eventual consistency) with the master by introducing the artificial delay on the secondary node. In this case, the master and secondary should temporarily return different lists of messages.

Add logic for messages deduplication and to guarantee the total ordering of messages.

**1. Оновлення master.py Додаємо підтримку параметра w, асинхронну реплікацію для демонстрації неконсистентності, дедуплікацію (за допомогою унікального ID повідомлення) і тотальний порядок:**

from flask import Flask, request, jsonify

import aiohttp

import asyncio

import logging

import time

import uuid

import threading

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']

async def replicate\_to\_secondary(secondary, message\_data, timeout=15):

    try:

        start\_time = time.time()

        logger.info(f"Attempting to replicate to {secondary}")

        async with aiohttp.ClientSession() as session:

            async with session.post(f"{secondary}/replicate", json=message\_data, timeout=timeout) as response:

                response.raise\_for\_status()

                logger.info(f"Received ACK from {secondary} in {time.time() - start\_time:.2f}s")

                return True

    except Exception as e:

        logger.error(f"Failed to replicate to {secondary}: {e}")

        return False

def run\_async\_tasks\_in\_background(tasks):

    loop = asyncio.new\_event\_loop()

    asyncio.set\_event\_loop(loop)

    try:

        loop.run\_until\_complete(asyncio.gather(\*tasks))

    finally:

        loop.close()

@app.route('/messages', methods=['POST'])

def post\_message():

    data = request.get\_json()

    if not data or 'message' not in data or 'w' not in data:

        logger.error("Invalid request: 'message' and 'w' fields are required")

        return jsonify({"error": "Message and write concern (w) are required"}), 400

    message = data['message']

    w = data['w']

    message\_id = str(uuid.uuid4())

    if not isinstance(w, int) or w < 1 or w > len(SECONDARIES) + 1:

        logger.error(f"Invalid write concern: w={w}, must be between 1 and {len(SECONDARIES) + 1}")

        return jsonify({"error": f"Write concern must be between 1 and {len(SECONDARIES) + 1}"}), 400

    message\_entry = {"id": message\_id, "message": message, "order": len(messages)}

    if any(m["id"] == message\_id for m in messages):

        logger.warning(f"Duplicate message ID {message\_id}, ignoring")

        return jsonify({"status": "Message already exists"}), 200

    messages.append(message\_entry)

    logger.info(f"Stored message: {message} with ID {message\_id}")

    tasks = [replicate\_to\_secondary(secondary, {"id": message\_id, "message": message, "order": message\_entry["order"]}) for secondary in SECONDARIES]

    if w == 1:

        threading.Thread(target=run\_async\_tasks\_in\_background, args=(tasks,), daemon=True).start()

        logger.info(f"Received 1 ACKs, satisfying w={w}")

        return jsonify({"status": "Message replicated", "message\_id": message\_id}), 200

    async def replicate\_with\_concern():

        acks = [True]

        required\_acks = w - 1

        if w == len(SECONDARIES) + 1:

            results = await asyncio.gather(\*tasks, return\_exceptions=True)

            for result in results:

                acks.append(result if isinstance(result, bool) and result else False)

            return acks

        for task in asyncio.as\_completed(tasks, timeout=15):

            try:

                result = await task

                acks.append(result if isinstance(result, bool) and result else False)

                if sum(acks) >= w:

                    return acks

            except Exception as e:

                logger.error(f"Task failed: {e}")

                acks.append(False)

                if sum(acks) >= w:

                    return acks

        return acks

    loop = asyncio.new\_event\_loop()

    asyncio.set\_event\_loop(loop)

    try:

        acks = loop.run\_until\_complete(replicate\_with\_concern())

    finally:

        loop.close()

    if sum(acks) >= w:

        logger.info(f"Received {sum(acks)} ACKs, satisfying w={w}")

        return jsonify({"status": "Message replicated", "message\_id": message\_id}), 200

    else:

        logger.error(f"Not enough ACKs: received {sum(acks)}, required w={w}")

        return jsonify({"error": "Not enough ACKs received"}), 500

@app.route('/messages', methods=['GET'])

def get\_messages():

    logger.info("Returning all messages")

    sorted\_messages = sorted(messages, key=lambda x: x["order"])

    return jsonify({"messages": [m["message"] for m in sorted\_messages]}), 200

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(host='0.0.0.0', port=5000)

#### 2. Оновлення secondary.py Додаємо підтримку дедуплікації та тотального порядку, а також затримку для емуляції неконсистентності:

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 'id' not in data or 'message' not in data or 'order' not in data:

        logger.error("Invalid request: 'id', 'message', and 'order' fields are required")

        return jsonify({"error": "ID, message, and order are required"}), 400

    message\_id = data['id']

    message = data['message']

    order = data['order']

    if any(m["id"] == message\_id for m in messages):

        logger.warning(f"Duplicate message ID {message\_id}, ignoring")

        return jsonify({"status": "ACK"}), 200

    time.sleep(5)

    messages.append({"id": message\_id, "message": message, "order": order})

    logger.info(f"Replicated message: {message} with ID {message\_id}")

    return jsonify({"status": "ACK"}), 200

@app.route('/messages', methods=['GET'])

def get\_messages():

    logger.info("Returning all messages")

    sorted\_messages = sorted(messages, key=lambda x: x["order"])

    return jsonify({"messages": [m["message"] for m in sorted\_messages]}), 200

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(host='0.0.0.0', port=5001)

**3. Оновлення client.py. Оскільки client.py виконує роль Secondary, оновлюємо його аналогічно до 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 'id' not in data or 'message' not in data or 'order' not in data:

        logger.error("Invalid request: 'id', 'message', and 'order' fields are required")

        return jsonify({"error": "ID, message, and order are required"}), 400

    message\_id = data['id']

    message = data['message']

    order = data['order']

    if any(m["id"] == message\_id for m in messages):

        logger.warning(f"Duplicate message ID {message\_id}, ignoring")

        return jsonify({"status": "ACK"}), 200

    time.sleep(8)

    messages.append({"id": message\_id, "message": message, "order": order})

    logger.info(f"Replicated message: {message} with ID {message\_id}")

    return jsonify({"status": "ACK"}), 200

@app.route('/messages', methods=['GET'])

def get\_messages():

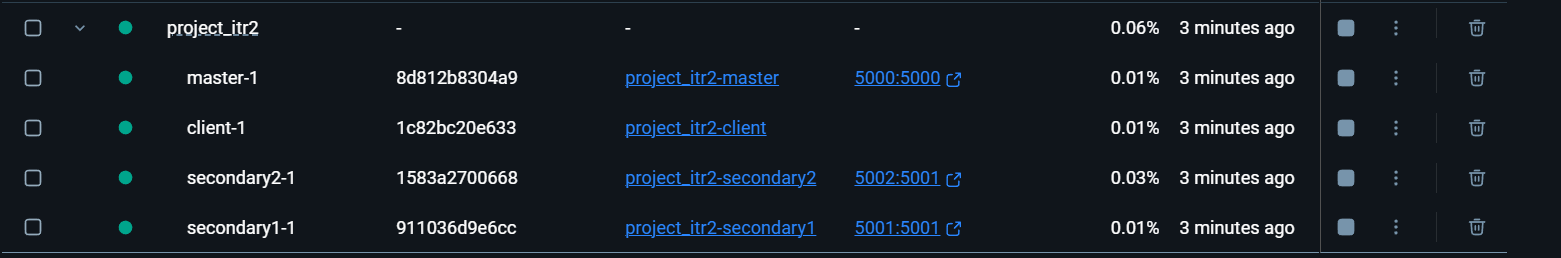
    logger.info("Returning all messages")

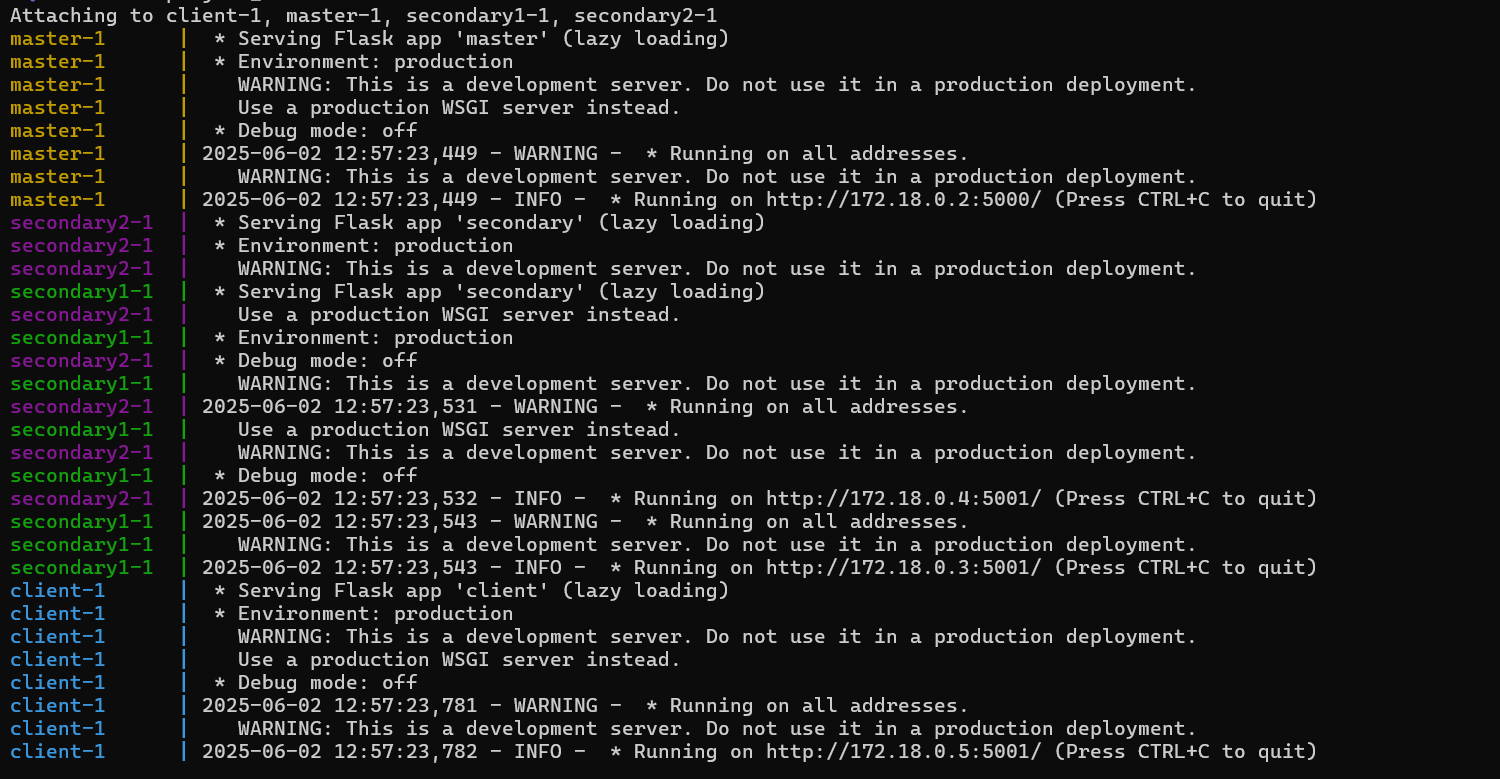
    sorted\_messages = sorted(messages, key=lambda x: x["order"])

    return jsonify({"messages": [m["message"] for m in sorted\_messages]}), 200

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(host='0.0.0.0', port=5001)

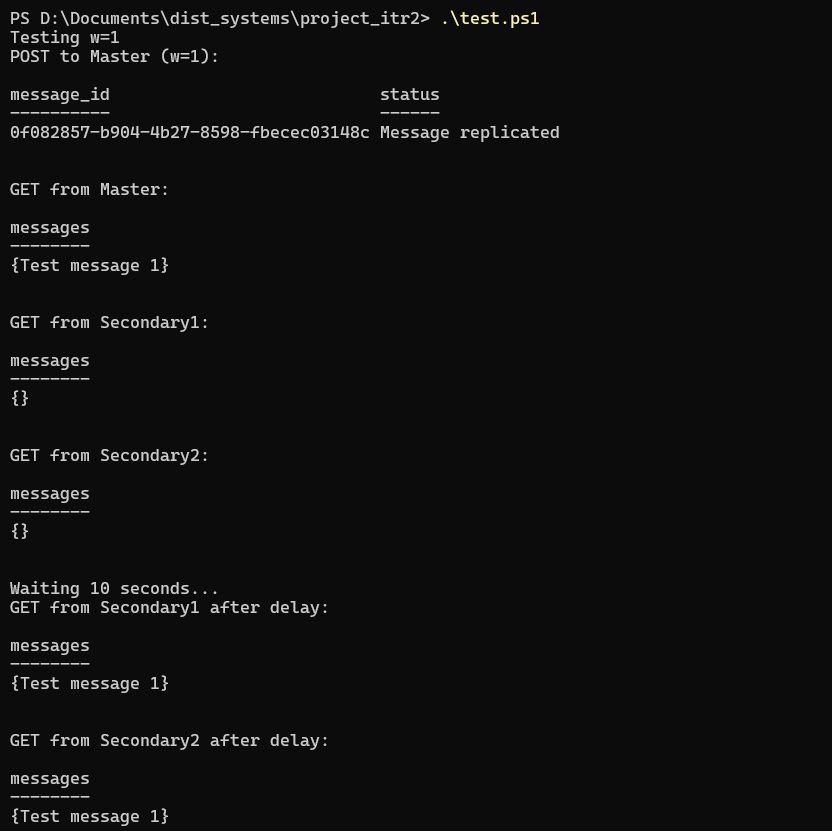




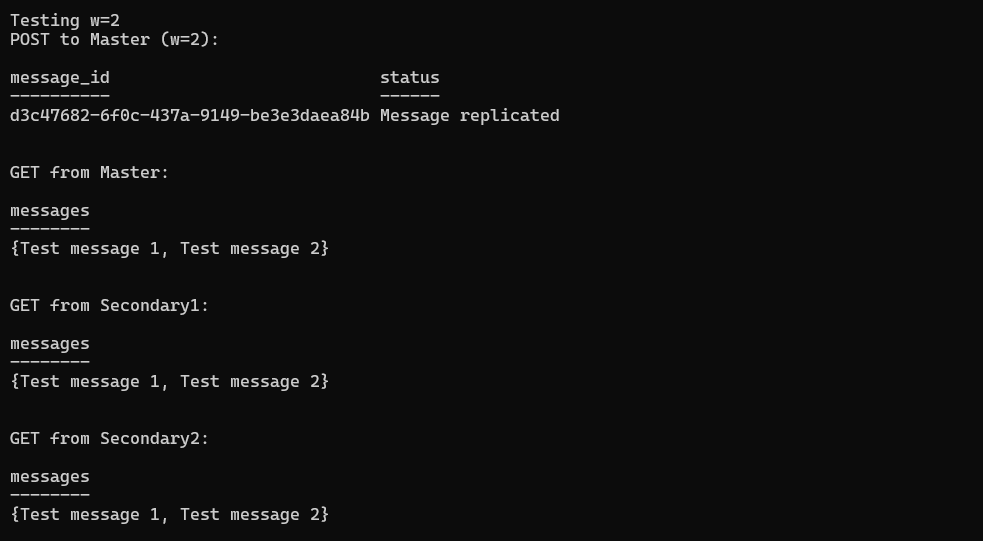
**Тестування POST-запиту з різними значеннями w**:

**W=1**

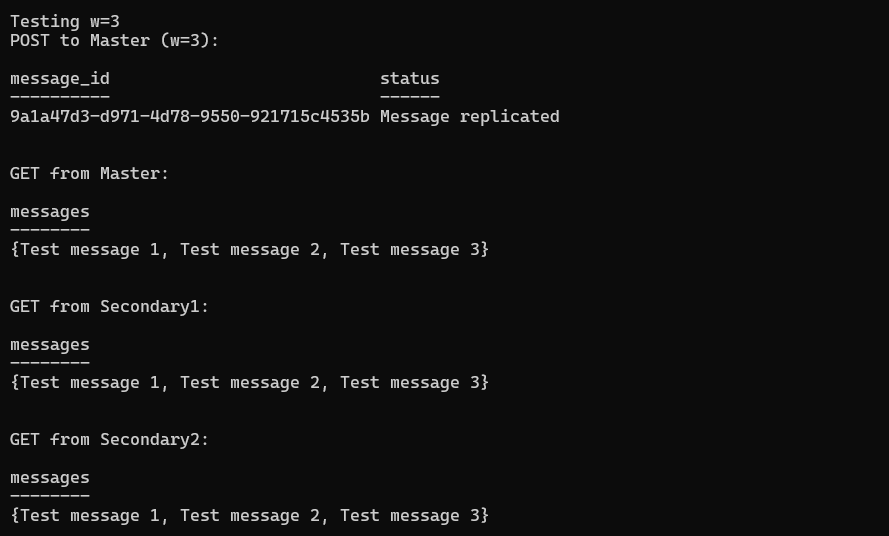
Демонструючи неконсистентність і остаточну консистентність



**W=2**

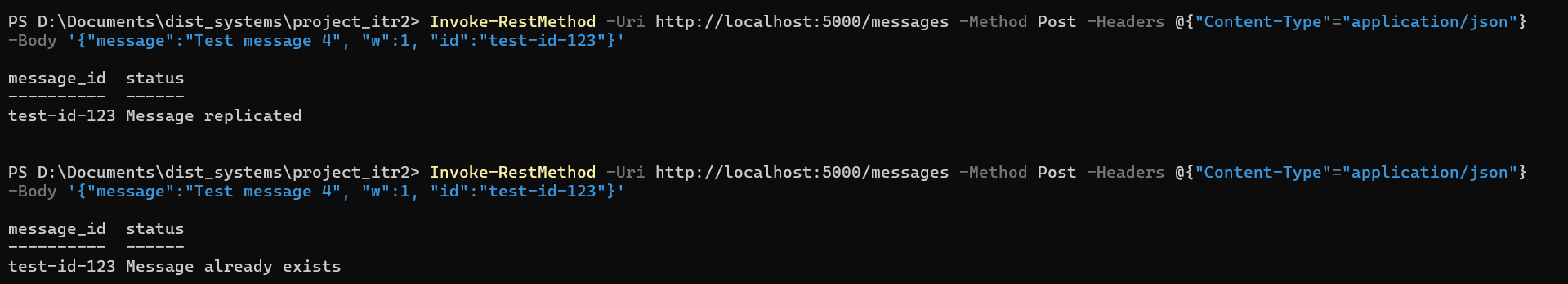


**W=3**

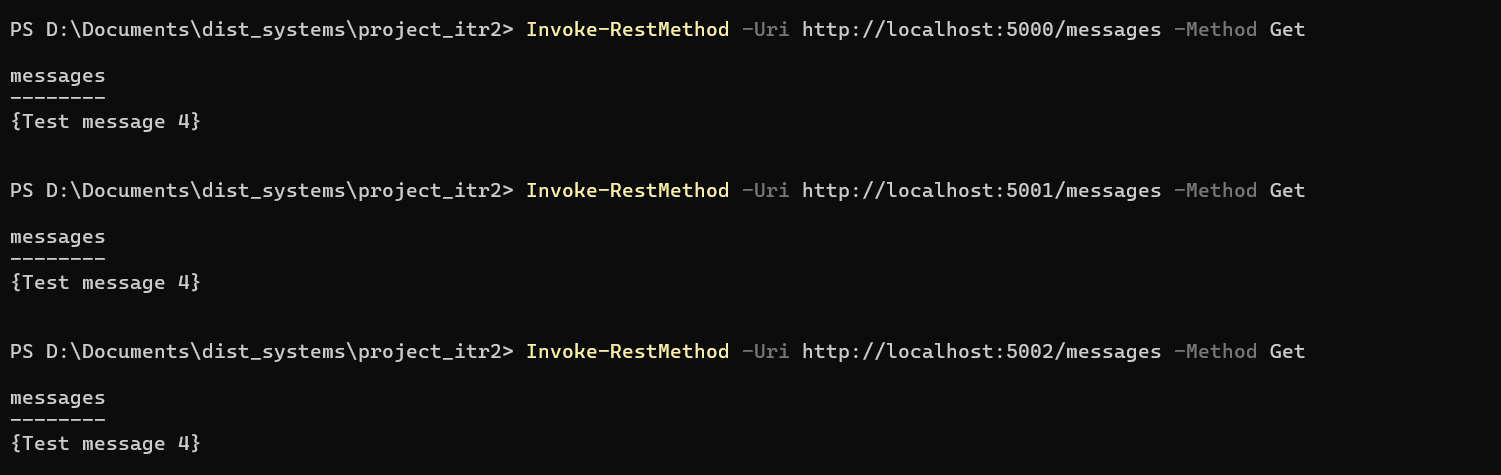
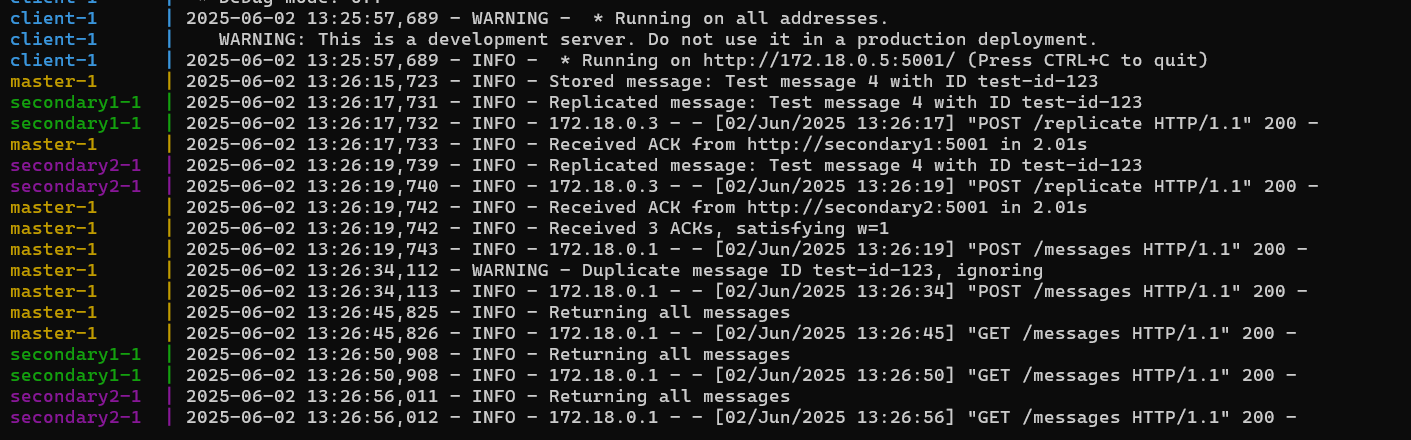


### 

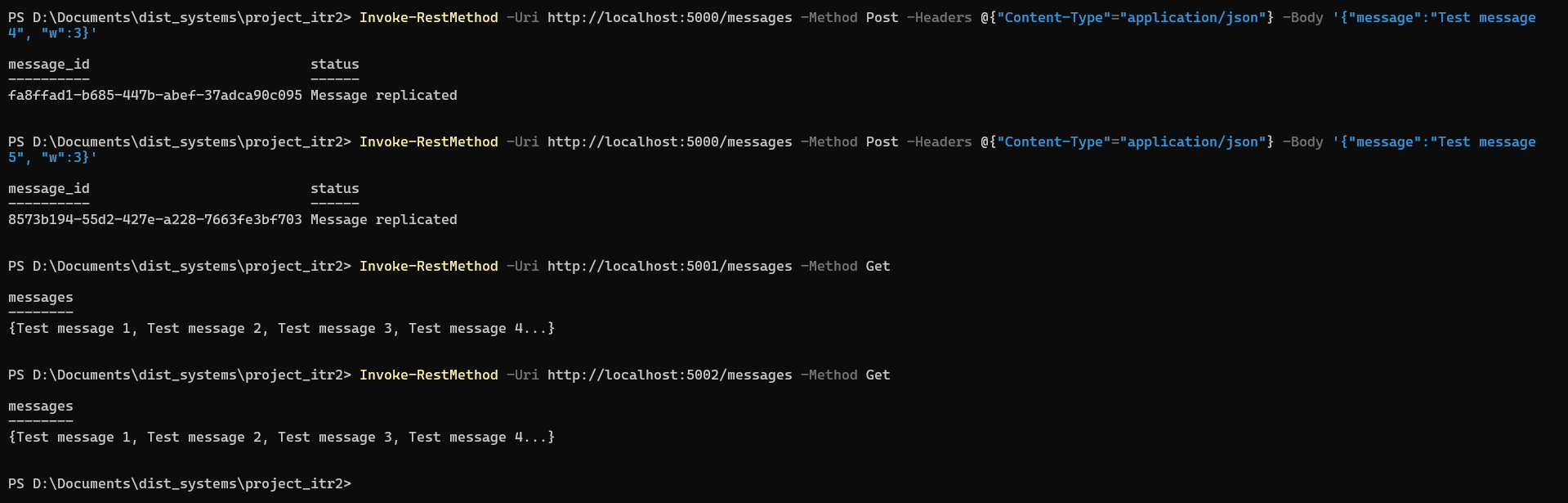
### **Тестування дедуплікації**

Тепер ми можемо надіслати два POST-запити з однаковим id, щоб перевірити, що Master виявляє дублювання і не додає повідомлення вдруге.

Виконали GET-запити, щоб переконатися, що повідомлення не дублюються:



**Тестування тотального порядку:**



Отож, було виконано напівсинхронну реплікацію з параметром w, емуляцію неконсистентності, дедуплікацію повідомлень також організували гарантію тотального порядку: поле order у повідомленнях і сортування в GET- запитах забезпечують одинаковий порядок.