**Iteration 1.**

* **5 points**

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

У docker-compose.yml визначено один сервіс master і два сервіси secondary1 та secondary2, що демонструє архітектуру з одним Master і кількома Secondary.

services:

  master:

    build: ./master

    ports:

      - "5000:5000"

      - "50050:50050"

    networks:

      - replicated-net

  secondary1:

    build: ./secondary

    ports:

      - "5001:5001"

      - "50051:50051"

    environment:

      - HTTP\_PORT=5001

      - GRPC\_PORT=50051

    networks:

      - replicated-net

  secondary2:

    build: ./secondary

    ports:

      - "5002:5002"

      - "50052:50052"

    environment:

      - HTTP\_PORT=5002

      - GRPC\_PORT=50052

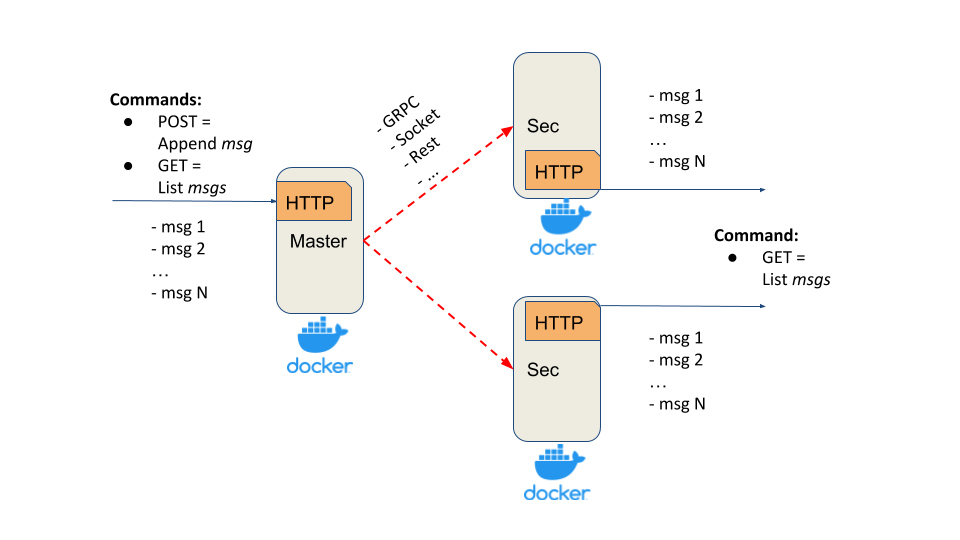
    networks:

      - replicated-net

networks:

  replicated-net:

    driver: bridge



***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

У master.py, рядок @app.route("/messages", methods=["POST"]) обробляє POST-запити. Рядок messages.append(message) додає повідомлення до списку messages.

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

def append\_message():

    message = flask.request.json.get("message")

    if not message:

        return flask.jsonify({"error": "No message provided"}), 400

    messages.append(message)

    log.info(f"Appending message: {message}")

    def replicate\_to\_secondary(addr):

        try:

            with grpc.insecure\_channel(addr) as channel:

                stub = replication\_pb2\_grpc.ReplicationServiceStub(channel)

                response = stub.ReplicateMessage(replication\_pb2.MessageRequest(message=message))

                return response.success

        except grpc.RpcError as e:

            log.error(f"Failed to replicate to {addr}: {e}")

            return False

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

if all(replicate\_to\_secondary(addr) for addr in secondary\_addresses):

        return flask.jsonify({"status": "success", "messages": messages}), 200

    else:

        return flask.jsonify({"error": "Failed to replicate to all secondaries"}), 500

***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

Secondary експонує простий HTTP-сервер через Flask із методом GET, який повертає всі репліковані повідомлення з in-memory списку messages.

Properties and assumptions:

* **after each POST request, the message should be replicated on every *Secondary* server**

У master.py, функція replicate\_to\_secondary надсилає повідомлення через gRPC до кожного Secondary, визначеного в secondary\_addresses. Кожен Secondary додає повідомлення до свого списку в ReplicateMessage.

* ***Master* should ensure that *Secondaries* have received a message via *ACK***

*У master.py, функція replicate\_to\_secondary повертає response.success, яке є ACK від Secondary. Ці ACK перевіряються в умові if all(replicate\_to\_secondary(addr) for addr in secondary\_addresses)*

* ***Master’s POST request* should be finished only after receiving *ACKs* from all *Secondaries* (blocking replication approach)**

У master.py, метод join() для потоків блокує виконання POST-запиту, доки всі потоки replicate\_to\_secondary не завершаться, тобто доки не отримано ACK від усіх Secondary.

* **to test that the replication is blocking, introduce the delay/sleep on the *Secondary***

У secondary.py, time.sleep(2), що додає затримку 2 секунди під час обробки кожного повідомлення в ReplicateMessage, що дозволяє перевірити блокуючу поведінку.

* **at this stage assume that the communication channel is a perfect link (no failures and messages lost)**

master.py логується помилка grpc.RpcError

* **any RPC framework can be used for *Master-Secondary* communication (Sockets, language-specific RPC, HTTP, Rest, gRPC, …)**

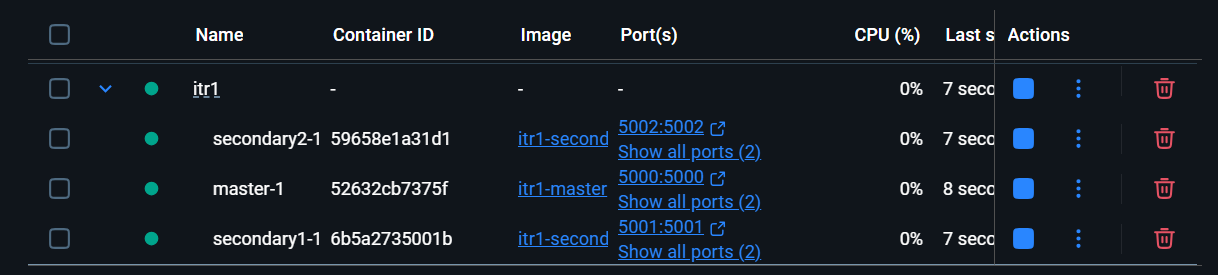
Використовується gRPC, визначений у replication.proto і реалізований через grpc у master.py і secondary.py.

* **your implementation should support logging**

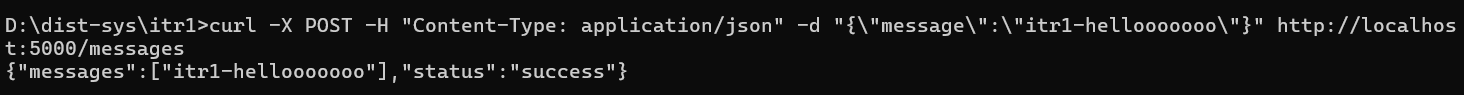
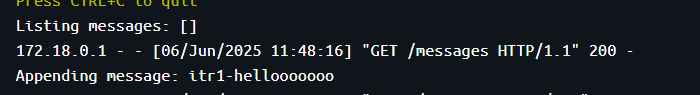
У master.py і secondary.py використовується модуль logging для запису подій

* ***Master* and *Secondaries* should run in Docker**

У docker-compose.yml усі сервіси (master, secondary1, secondary2) визначаються з використанням build, що вказує на запуск у Docker-контейнерах



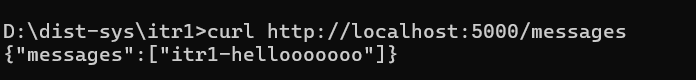
Додаємо повідомлення (POST до Master):

Запит заблокується на ~2 секунди через затримку в Secondary (time.sleep(2)), що підтверджує блокуючу реплікацію.

Отримання список повідомлень (GET):

З Master:



З Secondary:

